Faculty of Sciences, Technology and Communication (FSTC)

Bachelor in Computer Science

SpyDroid

Student ID: 0090551807
Student Name: Ferreira Coelho Luis Filipe
Email: luis.ferreira.002@student.uni.lu

Local Supervisor:
Dr. Ir. Hugo Jonker

Academic Supervisor:
Prof. Dr. Sjouke Mauw

Local Establishement:
University of Luxembourg
Faculty of Sciences, Technology and Communication (FSTC)
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg

December 28, 2012

Summary

English

This document aims to describe the context and the whole process I took with this research project done in the university. In addition it explains what path I decide to take to achieve it.

French

Ce document a pour but de décrire le contexte et toute la démarche que j'ai dû entreprendre pour réaliser ce projet de recherche a l'université. De plus il explique quel solution j'ai décidé d'utiliser pour le réaliser.

General Terms

security, information

Keywords

smartphones, mobile phones, sensors, accelerometer, android, information leakage, security

Abstract

Nowadays cell phones are more and more equipped with new and modern sensors, ranging from a 3-axis accelerometer to a barometic pressure sensor. These sensors enable phones to react to their environment, improving usability. They are able to help us with ease with our daily life and today most people can't imagine a life without their phone. But what if those "smartphones" with their impressive range of sensors is used to spy on its user? The goal of this thesisis to apply the work of Verma et al. on turning Apple's iPhone into a keyboard spying device to the popular Android framework. We show that: 1) a stand-alone application, running only on the phone, lacks the processing power to duplicate their results; 2) that the rate at which sensor data is made available via the Android framework is insufficient to duplicate these results; 3) that this rate cannot be improved by going to the lower JNI layer; 4) that even accessing the linux kernel directly also not possible is.

Declaration of Honesty

I, the undersigned , state that this thesis is entirely my work and no part has been copied from another person or from other sources.

Signature

Appreciation

Hereby i want to thank some people for their support in my Bachelor Thesis

For Yann le Corre

Just wanted to write you a note to thank you very much for your assistance in correcting and helping me with this project and some of my problems. Problems that would take me hours you solved it in minutes. It certainly was a pleasure to have someone respond so quickly to correct and help me with my problems. Many thanks

For Hugo Jonker

Thank you for your help and support for everything and the guidance that i needed. Thank you for motivating me when they were some technical problems. And thank you for our breaks were we could discuss and brainstorm. This really helped me a lot. Many thanks

My family friends

Thank you for supporting me in every possible way.

I dedicate this paper especially to my grand mother.

Signature

Contents

Chapter 1

Introduction

1.1 Introduction

In this paper we are going to use an android powered phone and then use his vast array of sensors for spying on the his owner. The phone is going to run a special software that when running and the owner is typing on a physical keyboard, is going to spy on him and try to guess what sentences he's typing whitout physical interaction.

1.2 Related Work

1.2.1 Keyboard Acoustic Emanations

In [Agr04] and [LZ05] they demonstrate that when typing on a keyboard the sound that is produced can be measured. And with those measurement we can guess what has been typed and then reproduce the typed sentences. [Agr04] is the original work and in [LZ05] is a better replicate of that work.

1.2.2 (sp)iPhone

In [AVT11] they do the exact same project but not with an Android powered smartphone but with an Iphone. An IOS device built and developped by Apple. In summary they have a sampling rate which is not enough for detect individual keystrokes. So they use probability and train a neural network to detect words directly by sensing the direction and seeing from which quadrant of the keyboard the keystroke comes from.

1.3 Approach

Our approach is that we are going to try the replicate project [AVT11] but in a way that it works for the Android plattform. We are going to try read the accelerometer data and then detect keystroke for guessing typed words.

Chapter 2

System Requirement

2.1 Equipment used

For this project we have used a total of three devices. Where we mainly focused on one of them. The other two where more used for checking if there were no device specific errors or problems. All three devices are developper friendly. They are devices developped and sold by Google in collaboration with Manufactures to mainly showcase what is possible with Android with the correct hardware. We used them because we have almost all the sources of them and they have powerful and modern hardware.

Here a quick summary of them:

Device	Device Nexus S (Google and Samsung) Phone
Operating System	Android 4.1.2 Jelly Bean
Processor	1 GHz Cortex A8 (Hummingbird)
Display	4.0” WVGA (480x800)
Memory	512 MB RAM (split 128MB GPU / 384MB OS)
Storage	16GB iNAND flash memory
Sensors	3-axis gyroscope Accelerometer Ambient light sensor Capacitive touch-sensitive buttons Digital compass Microphone Multi-touch capacitive touchscreen Proximity sensor Push buttons
Connectivity	GSM: 850, 900, 1800, 1900 HSDPA (7.2Mbps) HSUPA (5.76Mbps) Assisted GPS (A-GPS) Wi-Fi: 802.11 n/b/g Bluetooth 2.1+EDR microUSB 2.0 Near Field Communication (NFC)
Dimensions	123.9 mm (4.88 in) H 63.0 mm (2.48 in) W 10.8 mm (0.43 in) D

Device	Galaxy Nexus (Google and Samsung) Phone
Operating System	Android 4.1.2 Jelly Bean (later upgraded to 4.2.1)
Processor	1.2 GHz dual-core ARM Cortex-A9
Display	4.65 in (118 mm) diagonal HD Super AMOLED 1280×720 px
Memory	1 GB
Storage	16GB iNAND flash memory
Sensors	Multi-touch capacitive touchscreen Accelerometer 3-axis gyroscope A-GPS Barometer 3-axis Digital compass Proximity sensor Dual microphones for active noise cancellation
Connectivity	GSM/GPRS/EDGE 850/900/1800/1900 HSPA 850/900/1700/1900/2100 HSDPA 21 Mbps HSUPA 5.76 Mbps 3.5 mm TRRS GPS DLNA Micro USB 2.0 with USB On-The-Go MHL Bluetooth NFC Wi-Fi 802.11a/b/g/n (2.4/5 GHz)
Dimensions	135.5 mm (5.33 in) H 67.94 mm (2.675 in) W 8.94 mm (0.352 in) D

Device	Nexus 7 (Google and Asus) Tablet
Operating System	Android 4.1.2 Jelly Bean (later upgraded 4.2.1)
Processor	1.3 GHz quad-core Cortex-A9 (T30L Tegra 3)
Display	7-inch (180 mm) diagonal IPS LCD 1280×800 px
Memory	1 GB
Storage	16GB flash memory
Sensors	Multi-touch capacitive touchscreen Accelerometer gyroscope proximity sensor digital compass GPS magnetometer microphone
Connectivity	0.14 in (3.5 mm) headphone jack Bluetooth 3.0 Wi-Fi (802.11 b/g/n @ 2.4 GHz) NFC Micro USB 2.0 docking pins
Dimensions	198.5×120×10.56 mm (7.81×4.7×0.416 in)

Then there is a backend that is also needed. We decided to use a Raspberry pi as webserver because of the portability and the fast and cheap replacement if something may go wrong.

Device	RaspberryPi
Operating System	Debian
CPU chline Memory	ARM1176JZF-S (armv6k) 1GHz MHz 512 MByte
Storage capacity	SD Card of 4GB
Ethernet	1 x 10/100
USB	2 x 2.0
Dimensions	85.60 mm × 53.98 mm (3.370 in × 2.125 in)

Also with this we got a Asus WL-330gE. This device is a mobile 150Mbit wireless N router but with a custom firmware called DD-WRT LINK also using it because of portability.

Device	Asus WL-330gE with custom firmware DD-WRT
Architecture	MIPS
Vendor	Asus
System-On-Chip	RT3050F
CPU Speed	320 Mhz
Flash-Chip	MX25L3205E
Flash size	4 MiB
Memory	32 MiB
Wireless	RT28xx
Ethernet	1 x 10/100
USB	1 x 2.0

2.2 Android 4.1.2

We are focussing and developping on the Android version 4.1.2 also called jelly bean. It's the latest iteration till now and using the most up to date API. Initial was planned to use 2.3 codename Gingerbread because most android powered phones are running this version but because of outdated version and the new Guidelines that google put in we felt that 4.1 was the most suitable for showcasing what's possible with this plattform.

2.3 Development Software

Eclipse with the android SDK and NDK for developing the phone application.
Octave is a open source numerical and graphical tool almost like Matlab.
Mysql Workbench for interacting and designing the database.

2.4 Server

The server is running debian with a lighttpd server. Decided to go with that for being light and not consume to much ressources. In addition there is a Mysql server running. Before that we had sqlite but because sqlite couldn't handle that much data we had to switch.

Chapter 3

Development

3.1 Environment development with SDK

3.1.1 Introduction

For the first approach we started to develop like any other application using the android sdk provided by Google. We wanted to use the new android 4.0 Guidelines. These are just design guidelines so nothing mandatory for this project but still wanted to comply to it. Basically we are going to collect an x amount of samples from the sensors while we tap on an wodden table to see if it's possible to read those claps just from sensor data.

3.1.2 Accelerometer

Before we start explain how the application work we have to give a little explanation about how accelerometer work. An accelerometer measures the phone acceleration forces. If it stands still the it will just measure the gravitational force of earth which is static. So when moving the phone it will sense an acceleration, also when vibrating. Which the latter is the one where we are interested. When tapping on a table, or in our case, typing on a keyboard, it is possible to sense the vibration that emancipate on the table.

In figure 3.2 we see three physical accelerometer for measuring the x,y,z gravitational forces like in 3.1. There is a spring and a metal ball in an casing that when moved the ball lags behind, that then stretches the spring. If we measure how much the spring stretches we can calculate the force gravity. In an electrical sensor there are various way to do the exact same thing. One way to do it is like an electromagnetic induction. It's almost the same

R0.5

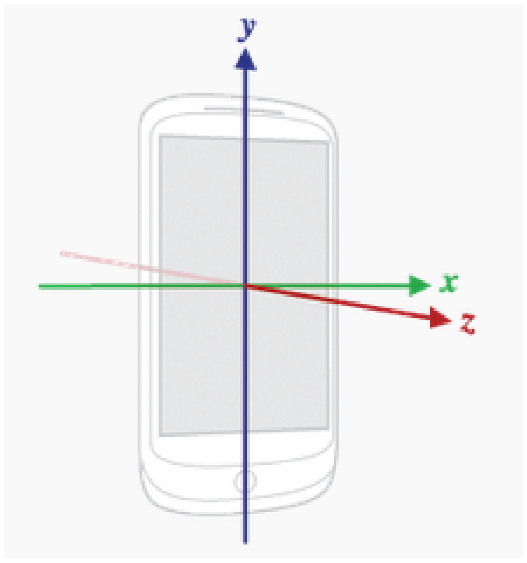


Figure 3.1: X Y Z Force measurement

L0.5

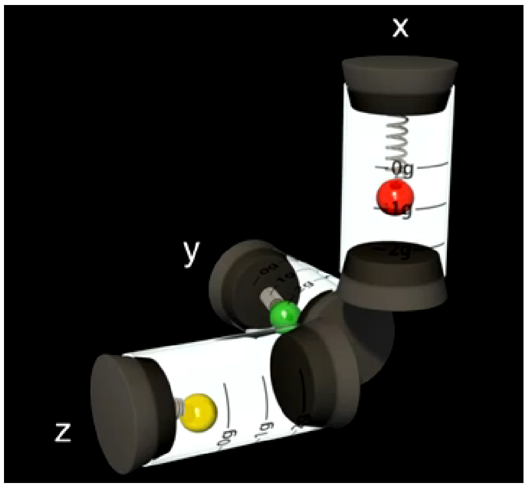


Figure 3.2: A Physical Accelerometer

r0.3

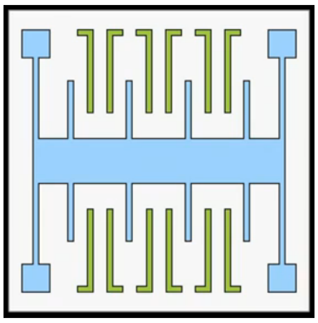


Figure 3.3: A digital Accelerometer found in mobile phones

like the physical one. We see in figure 3.3 there is again an housing. The blue part is the seismic mass that when a force occurs this part will move. Now if it moves, the fingers (green part) will generate an inductive current accordingly which we then can measure and see how much force has been produced.

3.1.3 Phone application in 4.1

By using the 4.0 developpment guidelines we have 4 fragments for each section of our application. We use a @link android.support.v4.app.FragmentPagerAdapter derivative, which will keep every loaded fragment in memory. This way we are sure that none goes into deep sleep. For normal application it is not one of the best solution because it is too memory intensive. For changing to the next fragment/section we only need to swipe left/right. But for our project it is ideal.
We have 4 fragments in total (figure 3.4). The first one (figure 3.5) is a simple description of the application. The second one (figure 3.6) shows us in real time the sensor data. The third one (figure 3.7) is a diagram of the sensor data. Here we use a simple open source library (figure [Geh11]) for showing the data on a line graph. And the forth one is for logging purpose. The whole application works in landscape and horizontal view (figure 3.8). And the last fragment is a log viewer for error check and debug purposes.

Fragment One "Main Section“

Each fragment consists of 3 simple textviewer. The top, middle and bottom viewer for better organization embedded in an simple table which is scrollable when it's too long. In fragment one the top viewer has the title. The middle

R0.5

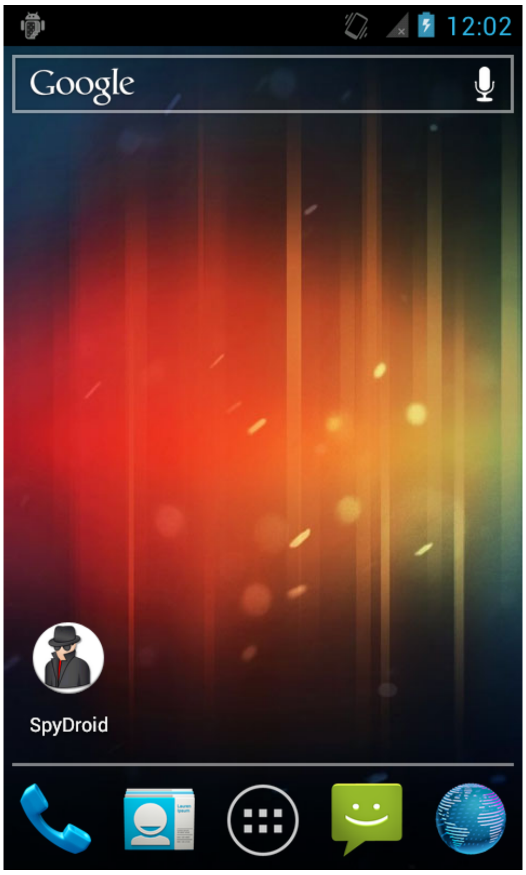
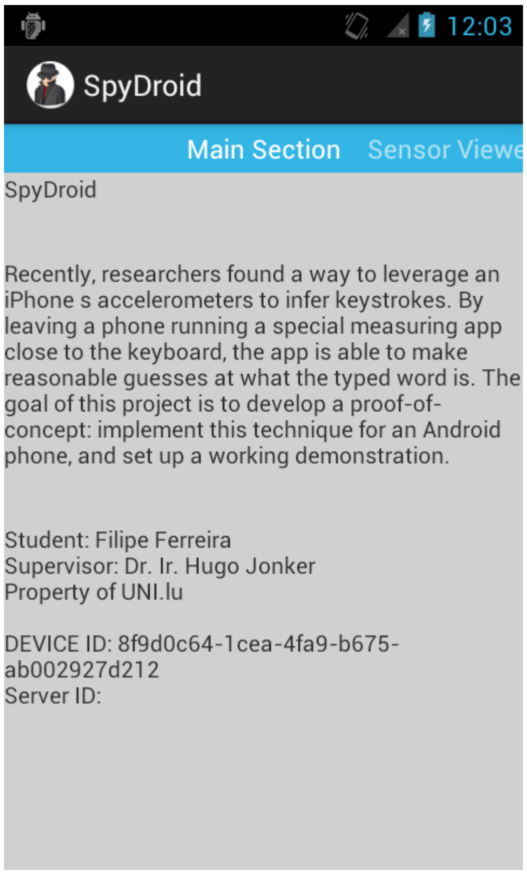


Figure 3.4: Phone main screen

R0.5



SpyDroid

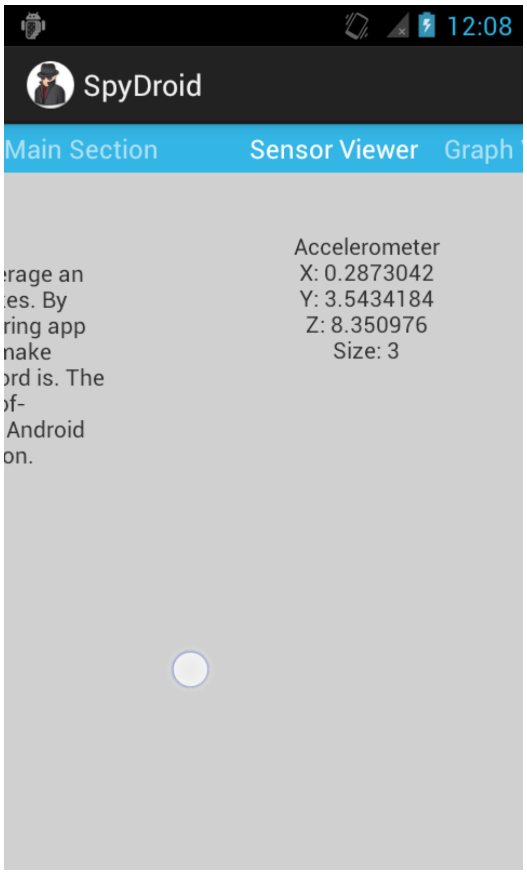
Recently, researchers found a way to leverage an iPhone s accelerometers to infer keystrokes. By leaving a phone running a special measuring app close to the keyboard, the app is able to make reasonable guesses at what the typed word is. The goal of this project is to develop a proof-of-concept: implement this technique for an Android phone, and set up a working demonstration.

Student: Filipe Ferreira
Supervisor: Dr. Ir. Hugo Jonker
Property of UNI.lu

DEVICE ID: 8f9d0c64-1cea-4fa9-b675-ab002927d212
Server ID:

Figure 3.5: Application description screen

R0.5



Accelerometer
X: 0.2873042
Y: 3.5434184
Z: 8.350976
Size: 3

erage an
tes. By
ring app
nake
ord is. The
of-
Android
on.

Figure 3.6: The sensor viewer while being swiped

R0.5

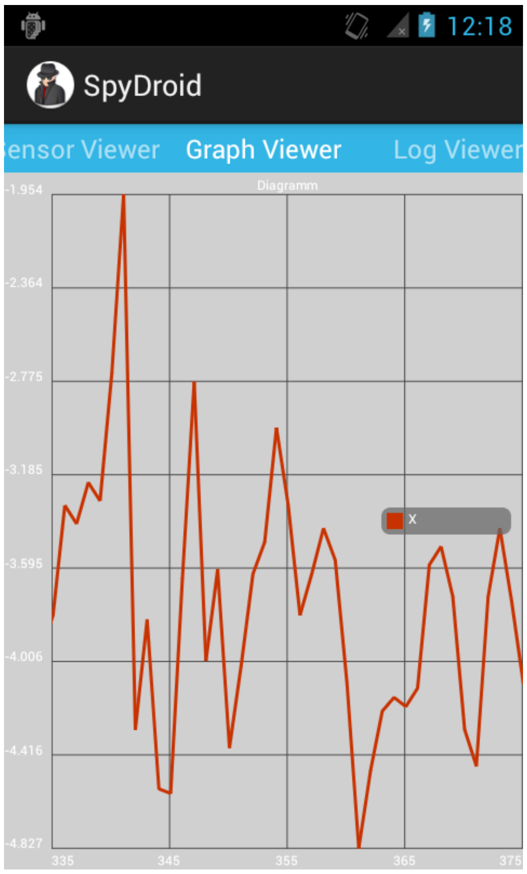


Figure 3.7: Graphical viewer

R0.5

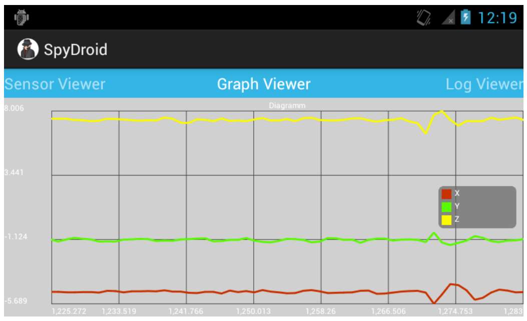


Figure 3.8: Graphical viewer 3 axis landscape

has the description. The bottom has an unique id of this device which has been generated on starting the application. And then we have a server id which shows, when connected to the server, the id of this device on the server.

Fragment One Code

```
      viewer =new ScrollView(getActivity());
      table =new TableLayout(getActivity());
      viewer.addView(table);

      TextView header = new TextView(getActivity());
      header.setGravity(Gravity.TOP);
      header.setText("");

      TextView center = new TextView(getActivity());
      center.setGravity(Gravity.MIDDLE);
      center.setText("");

      TextView bottom = new TextView(getActivity());
      bottom.setGravity(Gravity.BOTTOM);
      bottom.setText("");

      table.addView(header);
      table.addView(center);
      table.addView(bottom);

      switch (args.getInt(ARG_SECTION_NUMBER)) {
case 1:// FRAGMENT 1
  center.setId(1);
  header.setText("SpyDroid\n\n");
  center.setText(
      "Recently, researchers found a way to leverage an iPhone s
      accelerometers to infer keystrokes. By leaving a phone
      running a special measuring app close to the keyboard,
      the app is able to make reasonable guesses at what the
      typed word is. The goal of this project is to develop a
      proof-of-concept: implement this technique for an
      Android phone, and set up a working demonstration.\n\n");
      bottom.setText("Student: Filipe Ferreira\n
      Supervisor: Dr. Ir. Hugo Jonker\n
      Property of UNI.lu\n\nDEVICE ID: "+uniqueID+"\n
      Server ID: "+serverId);
  break;
case 2......//FRAGMENT 2
```

Unique ID generation Android natively doesn't have a unique ID generator for devices that is reliable. And we need one for identifying the devices on the backend.

```
  private static String uniqueID = null;
  private static final String PREF_UNIQUE_ID = "PREF_UNIQUE_ID";

  public synchronized static String id(Context context) {
      if (uniqueID == null) {
          SharedPreferences sharedPrefs =
          context.getSharedPreferences(The sensor viewer while being swiped
                  PREF_UNIQUE_ID, Context.MODE_PRIVATE);
          uniqueID = sharedPrefs.getString(PREF_UNIQUE_ID, null);
          if (uniqueID == null) {
              uniqueID = UUID.randomUUID().toString();
              Editor editor = sharedPrefs.edit();
              editor.putString(PREF_UNIQUE_ID, uniqueID);
              editor.commit();
          }
      }
      return uniqueID;
```

Fragment Two "Sensor viewer"

Again here we have three texbox for better organization. When an sensor change event occurs the value is going to be submitted real time in those texboxes.

Sensor initialization Code When the application starts we directly initialize the sensor with a listener so that when a sensor change event triggers we can update our textbox and later send those values to the server. Also we have to pay attention to put this code snippet on the onresume, onpause,.. event so that it doesn't stop when the screen turns off.

```
      sensor=(SensorManager)getSystemService(SENSOR_SERVICE);
      sensor.registerListener(this,
      sensor.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
      SensorManager.SENSOR_DELAY_FASTEST );

      sensor.registerListener(this,
      sensor.getDefaultSensor(Sensor.TYPE_GYROSCOPE),
      SensorManager.SENSOR_DELAY_FASTEST );

      sensor.registerListener(
      this,
      sensor.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
      SensorManager.SENSOR_DELAY_FASTEST );
```

Sensor Change Event Code Here we have to pay attention which sensor has triggered this event. And if the sensor that we want to read has triggered the event we can then read our values and show them onscreen.

```
if(event.sensor.getType()==Sensor.TYPE_ACCELEROMETER){

      float x=event.values[0];
      float y=event.values[1];
      float z=event.values[2];

      TextView center =(TextView)findViewById(2);
      if(center!=null){
      header.setText("Real Time Sensors");
      center.setText(
  "\nAccelerometer\n" +
  "X: "+x+
  "\nY: "+y+
  "\nZ: "+z+"\nSize: "
  event.values.length+"\n");
}
```

Fragment Three ”Graph Viewer“

Here we are using an open source library [Geh11]. We then can visualize the 3 axis directly on the phone.

Graph viewer initialization The graph viewer is been drawn on the fragment page directly. This way we are sure that there are already sensor data stored in our global array for the axis which are called xSeries, ySeries, zSeries.

```
GraphViewData[] data = new GraphViewData[1];

xSeries = new GraphViewSeries("X",
  new GraphViewStyle(Color.rgb(200, 50, 00), 3),data);
ySeries = new GraphViewSeries("Y",
  new GraphViewStyle(Color.rgb(90, 250, 00), 3),data);
zSeries = new GraphViewSeries("Z",
  new GraphViewStyle(Color.rgb(250, 250, 00), 3),data);

GraphView graphView;
graphView = new LineGraphView(this.getActivity(),"Diagramm");

graphView.addSeries(xSeries);
graphView.addSeries(ySeries);
graphView.addSeries(zSeries);

graphView.setShowLegend(true);
graphView.setScrollable(true);
graphView.setScalable(true);
graphView.setHorizontalScrollBarEnabled(true);
graphView.setVerticalScrollBarEnabled(true);
graphView.setId(4);
LinearLayout viewer1 =new LinearLayout(getActivity());
viewer1.addView(graphView);
graphView.setViewPort(2, 40);
return viewer1;
```

The code snippet where we are storing our axis values in the on sensor change event.

```
    if(xSeries!=null && ySeries!=null && zSeries!=null){
    posSeries++;
    xSeries.appendData(new GraphViewData(posSeries, x), true);
    ySeries.appendData(new GraphViewData(posSeries, y), true);
    zSeries.appendData(new GraphViewData(posSeries, z), true);
```

```
GraphView graph =(GraphView)findViewById(4);
graph.redrawAll();
}
```

Fragment four ”Log viewer“

In this fragment we only have one textbox were it’s been populated with our system debug output coming from logcat which is android’s own logging system.

Fragment Three ”Graph Viewer“ Problem

Unfortunately here we can see that’s not a viable solution. First the screen is too small to visualize it so that we have to zoom in and out the whole time. Also the phone is too underpowered for that much information. So this is why we decided to go for a backend where we are going to send our values where they are going to be stored and then we can visualize

Send the values to a mysql server

We decided for a mysql server because it’s good solution for handle a large amount of data in a small amount of time and that doesn’t use much memory on runtime. Because android new 4.0 api also changes how it send data to a sql server. It has to spawn a new thread when making a new http post to a webserver. This means that we have to be careful how we send the data to the server. We can’t spawn a thread for each event that occurs. It would be too much threads that would wait in our pool to be threated. So what we are going to do is collect an amount of 200 values for each x,y,z value and then send that packet. That way we send 600 values in total in each packet with their respective timestamp. Before that we also register the device so that we can identify from which device those values comes. In figure 3.9 we see the database structure.

Registering When starting the application he’s going to check if the server is on. If it is then he’s going to check if his id already exists in the server. If yes then he’s going to send directly packages. If it doesn’t exist he’s going to register itself on the server.
Here is the application’s code snippet

```
//registering the device
 GetServerId getServerid=new GetServerId(uniqueID);
```

R1

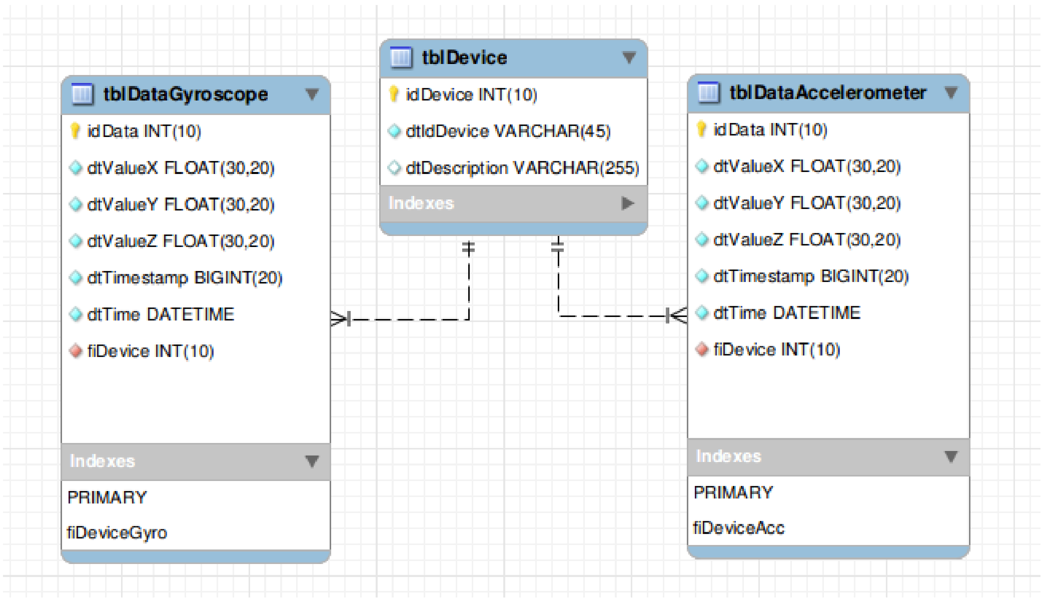
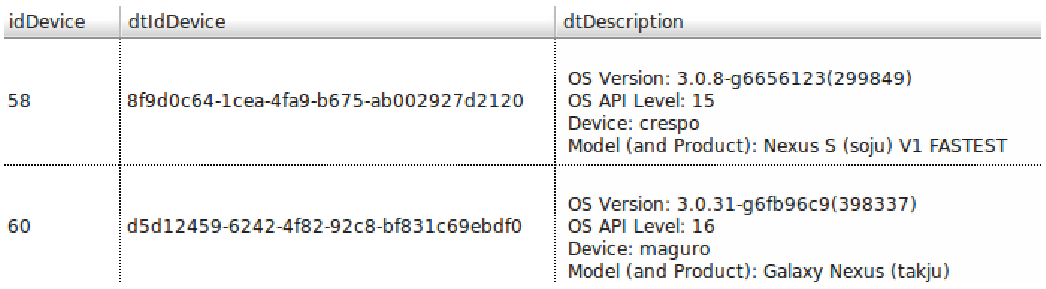


Figure 3.9: Our database structure

```
        getServerid.execute();

        if(serverId.isEmpty()){
            getServerid=new GetServerId(uniqueID);
            getServerid.execute();

        }

        //Register class
      private class GetServerId extends AsyncTask<URL, Integer, Long> {
      private String phId;

      public GetServerId (String phoneId){
if(phoneId!=null)
  phId=phoneId;
}

      protected Long doInBackground(URL... urls) {
InputStream is = null;
```

```
String result = "";
ArrayList<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
nameValuePairs.add(new BasicNameValuePair("varId",phId));

String desc="\n OS Version: " + System.getProperty("os.version")
+ "(" + android.os.Build.VERSION.INCREMENTAL + ")";
desc += "\n OS API Level: " + android.os.Build.VERSION.SDK;
desc += "\n Device: " + android.os.Build.DEVICE;
desc += "\n Model (and Product): " + android.os.Build.MODEL
+ " ("+ android.os.Build.PRODUCT + ")";

nameValuePairs.add(new BasicNameValuePair("varDesc",desc));

try{
  HttpClient httpclient = new DefaultHttpClient();
  HttpPost httppost = new HttpPost("http://192.168.1.100/insert.php");
  httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs, HTTP.UTF_8));
  HttpResponse response = httpclient.execute(httppost);
  HttpEntity entity = response.getEntity();
  is = entity.getContent();

}
catch(Exception e)
{

}

String returnString="";

try{
  StringBuilder sb=null;
  BufferedReader reader = new BufferedReader(
    new InputStreamReader(is,"iso-8859-1"),8);
  sb = new StringBuilder();
  sb.append(reader.readLine() + "\n");

  String line="0";
  while ((line = reader.readLine()) != null) {
sb.append(line + "\n");
  }
  is.close();
```

R1



idDevice	dtIdDevice	dtDescription
58	8f9d0c64-1cea-4fa9-b675-ab002927d2120	OS Version: 3.0.8-g6656123(299849) OS API Level: 15 Device: crespo Model (and Product): Nexus S (soju) V1 FASTEST
60	d5d12459-6242-4f82-92c8-bf831c69ebdf0	OS Version: 3.0.31-g6fb96c9(398337) OS API Level: 16 Device: maguro Model (and Product): Galaxy Nexus (takju)

Figure 3.10: Two different devices in our database

```
  result=sb.toString();
  returnString=result.substring(result.lastIndexOf("{\"id\":\"")
    +7, result.indexOf("\"}"));
  }catch(Exception e){

  }
serverId=returnString;

return (long) 0;
      }

      protected void onProgressUpdate(Integer... progress) {
setProgressPercent(progress[0]);
      }

      protected void onPostExecute(Long result) {
showDialog("Downloaded " + result + " bytes");
      }

      }
```

Here the webserver's code snippet for registering the device. Before inserting the values he's going to check if this device exists because we need his id for identifying the values between all the devices. In figure 3.10 we can see how it's going to appear in our database.

```
<?php
        $con = mysql_connect("localhost","spyuser","spydroidpassw0rd");
        if (!$con){
                die('Could not connect: ' . mysql_error());
        }

        mysql_select_db("spydroiddb", $con);

if(isset($_POST['varId']) && !empty($_POST['varId'])){
$varID=$_POST['varId'];
$sql2="SELECT idDevice from tblDevice where dtIdDevice='$varID';";

$result=mysql_query($sql2);
if (mysql_num_rows($result)!=0){
$row = mysql_fetch_assoc($result);
$out['id']=$row['idDevice'];
print json_encode($out);

}else {
$varDesc=$_POST['varDesc'];
$sql2="INSERT INTO tblDevice
  (dtIdDevice,dtDescription) VALUES ('$varID','$varDesc');";
if (!mysql_query($sql2,$con)){
        die('Error: ' . mysql_error());
}

$out['id']=mysql_insert_id();
print json_encode($out);
}
}
```

Now that we have an id for this device on the server we can send our values to the server. The package consists of the 200 values for each axis separated by ";“ for parsing purposes with each timestamp of that value. We collect 200 events and then attach our id for identifying and put in our thread pool for sending that packet. After that reset our array for the next packet.

```
  //code snippet in our sensor change event for collecting our values
  if(nArrA==200){
```

```
  ArrayList<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
  nameValuePairs.add(new BasicNameValuePair("valXA", xArrA));
  nameValuePairs.add(new BasicNameValuePair("valYA", yArrA));
  nameValuePairs.add(new BasicNameValuePair("valZA", zArrA));
  nameValuePairs.add(new BasicNameValuePair("timeA", tArrA));
  nameValuePairs.add(new BasicNameValuePair("varAId", serverId));

  SendToServer toSend=new SendToServer(nameValuePairs);
  toSend.execute();

  xArrA="";
  yArrA="";
  zArrA="";
  tArrA="";
  nArrA=0;
  }

  xArrA+=x+";";
  yArrA+=y+";";
  zArrA+=z+";";
  tArrA+=event.timestamp+";";
  nArrA++;



  //our function for sending our packet to the server
private class SendToServer extends AsyncTask<URL, Integer, Long> {
private ArrayList<NameValuePair> nameValuePairs;

    public SendToServer (ArrayList<NameValuePair> sendData){
if(sendData!=null)
      nameValuePairs=sendData;
}

    protected Long doInBackground(URL... urls) {
      try {
  HttpClient httpclient = new DefaultHttpClient();
  HttpPost httppost = new HttpPost("http://192.168.1.100/insert.php");
  // Execute HTTP Post Request
  HttpResponse response = httpclient.execute(httppost);
```

```
      } catch (ClientProtocolException e) {
  center1.append("\n"+e+"\n");

      } catch (IOException e) {
  center1.append("\n"+e+"\n");
      }

      return (long) 0;
    }
}
```

Because our application needs now a network connection we have to add to the android manifest that we need the internet permission.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.uni.lu.spydroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main"
            android:configChanges="keyboardHidden|orientation|screenSize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Now on the server part we have to parse the packet and then save the values to the database. This is done in the insert.php file. After we have the check the id of the device we parse and save the values.

```
if(isset($_POST['valXA']) && !empty($_POST['valXA'])){

$arr1A=explode(";",$_POST['valXA']);
$arr2A=explode(";",$_POST['valYA']);
$arr3A=explode(";",$_POST['valZA']);
$arr4A=explode(";",$_POST['timeA']);

$varID=$_POST['varAId'];
$n=count($arr1A);

for($i=0;$i<$n-1;$i++) {
      $var1A=$arr1A[$i];
      $var2A=$arr2A[$i];
      $var3A=$arr3A[$i];
      $var4A=$arr4A[$i];

      $sql="INSERT INTO tblDataAccelerometer
(dtValueX, dtValueY, dtValueZ, dtTimestamp,dtTime,fiDevice)
VALUES ('$var1A','$var2A','$var3A','$var4A',NOW(),'$varID');";
      if (!mysql_query($sql,$con)){
      die('Error: ' . mysql_error());
      }
}
```

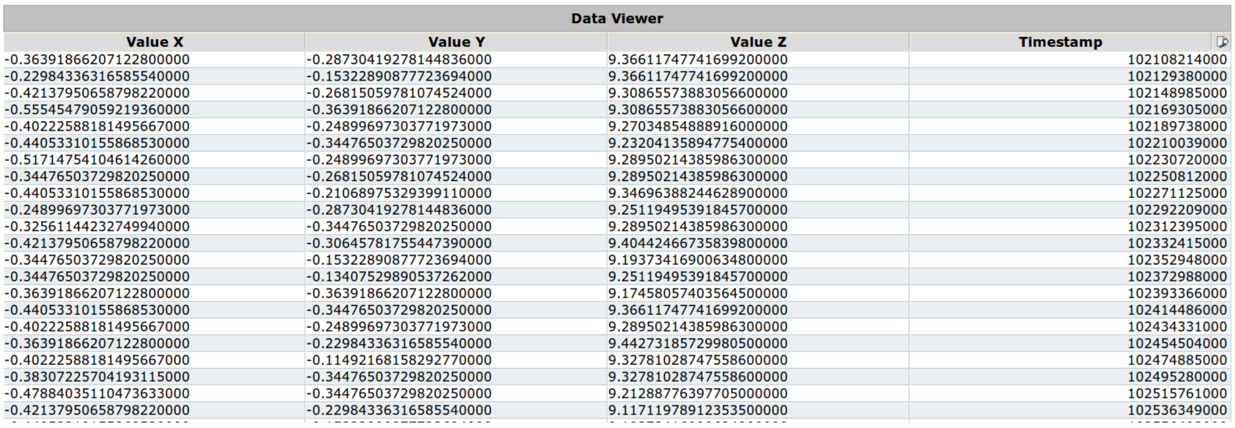
3.1.4 Server application

Now that our values are saved. We have to check how we are going to handle everything on the server. Here two open source library are being used one for showing the values in tables [Sis07] and the other one for showing in graphs [AS12]. The webapplication basically conists of these two plus a page where some calcuations are done like the average, sums,...

Here the data grid view where we can view our values.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Data Viewer</title>
<?php
    include ("phpmydatagrid.class.php");
    $dataViewer = new datagrid;
    $dataViewer -> friendlyHTML();
    $dataViewer -> pathtoimages("./images/");
    $dataViewer -> closeTags(true);
    $dataViewer -> form('data', true);
    $dataViewer -> methodForm("post");
    $dataViewer -> searchby("idData,dtValueX,dtValueY,dtValueZ,dtTimestamp");
    $dataViewer -> decimalDigits(100);
    $dataViewer -> decimalPoint(".");
    $dataViewer -> conectadb
      ("localhost", "spyuser", "spydroidpassw0rd" , "spydroiddb");
    $dataViewer -> tabla ("tblDataAccelerometer");
    $dataViewer -> TituloGrid("Data Viewer");
    $dataViewer -> FooterGrid
      ("Property of Uni.lu\nCopyright © Université du Luxembourg 2012.
All rights reserved");
    $dataViewer -> datarows(500);
    $dataViewer -> paginationmode('links');
    $dataViewer -> orderby("idData", "ASC");
    $dataViewer -> noorderarrows();

    $dataViewer -> FormatColumn
      ("dtValueX", "Value X", 0, 0, 0, "100", "left");
    $dataViewer -> FormatColumn
      ("dtValueY", "Value Y", 30, 30, 0, "100","left");
    $dataViewer -> FormatColumn
      ("dtValueZ", "Value Z", 30, 30, 0, "100", "left");
    $dataViewer -> FormatColumn
      ("dtTimestamp", "Timestamp", 25, 25, 0, "100", "right");

    $dataViewer -> setHeader();
    echo "</head><body>";
    $dataViewer -> grid();
```

R1.2



Data Viewer

Value X	Value Y	Value Z	Timestamp
-0.363918662071228000000	-0.287304192781448360000	9.366117477416992000000	102108214000
-0.229843363165855400000	-0.153228908777236940000	9.366117477416992000000	102129380000
-0.421379506587982200000	-0.268150597810745240000	9.308655738830566000000	102148985000
-0.555454790592193600000	-0.363918662071228000000	9.308655738830566000000	102169305000
-0.402225881814956670000	-0.248996973037719730000	9.270348548889160000000	102189738000
-0.440533101558685300000	-0.344765037298202500000	9.232041358947754000000	102210039000
-0.517147541046142600000	-0.248996973037719730000	9.289502143859863000000	102230720000
-0.344765037298202500000	-0.268150597810745240000	9.289502143859863000000	102250812000
-0.440533101558685300000	-0.210689753293991100000	9.346963882446289000000	102271125000
-0.248996973037719730000	-0.287304192781448360000	9.251194953918457000000	102292209000
-0.325611442327499400000	-0.344765037298202500000	9.289502143859863000000	102312395000
-0.421379506587982200000	-0.306457817554473900000	9.404424667358398000000	102332415000
-0.344765037298202500000	-0.153228908777236940000	9.193734169006348000000	102352948000
-0.344765037298202500000	-0.134075298905372620000	9.251194953918457000000	102372988000
-0.363918662071228000000	-0.363918662071228000000	9.174580574035645000000	102393366000
-0.440533101558685300000	-0.344765037298202500000	9.366117477416992000000	102414486000
-0.402225881814956670000	-0.248996973037719730000	9.289502143859863000000	102434331000
-0.363918662071228000000	-0.229843363165855400000	9.442731857299805000000	102454504000
-0.402225881814956670000	-0.114921681582927700000	9.327810287475586000000	102474885000
-0.383072257041931150000	-0.344765037298202500000	9.327810287475586000000	102495280000
-0.478840351104736330000	-0.344765037298202500000	9.212887763977050000000	102515761000
-0.421379506587982200000	-0.229843363165855400000	9.117119789123535000000	102536349000

Figure 3.11: Our data showing in the webserver

```
    $dataViewer -> desconectar();
?>
</body>
</html>
```

And in figure 3.11 is how it looks like.

For putting the data in an graph we first have to query the values. And those values stored in array have to been painted in an canvas with the javascript library.

```
        //fetching the data
        $con = mysql_connect("localhost","spyuser","spydroidpassw0rd");
        if(!$con){
die('Could not connect: ' . mysql_error());
        }

        mysql_select_db("spydroiddb", $con);

        $resultAccelerometer = mysql_query("SELECT * FROM tblDataAccelerometer;");

        $xA;
        $yA;
        $zA;
```

```
    $nA=-1;
    while($row = mysql_fetch_array($resultAccelerometer)){
$nA++;
$xA[$nA]=$row[1];
$yA[$nA]=$row[2];
$zA[$nA]=$row[3];
    }

    //show it in our graph
  $(function () {
  $(document).ready(function() {
      chartAX = new Highcharts.Chart({
          chart: {
              renderTo: 'containerXA',

              zoomType: 'x',
              spacingRight: 20/*,
              events: {
                  load: function() {

                      // set up the updating of the chart each second
                      var series = this.series[0];
                      setInterval(function() {
                          var x = (new Date()).getTime(), // current time
                              y = Math.random();
                          series.addPoint([x, y]
                          , true, true);
                      }    $(document).ready(function() {
      chartAX = new Highcharts.Chart({
          chart: {
              renderTo: 'containerXA',

              zoomType: 'x',
              spacingRight: 20
          },
          title: {
              text: 'Data Viewer Accelerometer X'
          },
          xAxis: {
```

```
        title: {
            text: null
        }
    },
    yAxis: {
        title: {
            text: 'Value'
        },

        startOnTick: false,
        showFirstLabel: false
    },
    tooltip: {
        shared: true
    },
    legend: {
        enabled: false
    },
    plotOptions: {
        area: {
            fillColor: {
                linearGradient: { x1: 0, y1: 0, x2: 0, y2: 1},
                stops: [
                    [0, Highcharts.getOptions().colors[0]],
                    [1, 'rgba(2,0,0,0)']
                ]
            },
            lineWidth: 1,
            marker: {
                enabled: false,
                states: {
                    hover: {
                        enabled: true,
                        radius: 5
                    }
                }
            },
            shadow: false,
            states: {
                hover: {
                    lineWidth: 1
```

```
                            }
                        }
                    }
                },

                series: [{
                    name: 'X',
                    data: <?php
$i=-1;
echo "[";
while($i<$nA){
$i++;
echo $xA[$i].", ";
}
echo $xA[$nA]."]"; ?>
                }]
            });
, 100);
</script>
</head>
<body>
<script src="js/highcharts.js"></script>
<script src="js/modules/exporting.js"></script>

<div id="containerXA" style="min-width: 400px; height: 400px; margin: 0 auto">
</div>
```

And in figure 3.12 we see 700 sample of the x-axis of our accelerometer

3.1.5 First Test run

After the setup we then finally could do some inital tests. We put the phone on a wooden table and then knocked on the table three times. We clearly could see those knocks (figure 3.13 unfiltered and 3.14 filtered.)

3.1.6 Conclusion and Solution

So now that the application works we could finally test on a keyboard. We first try with an office dell keyboard with elevated stand. Because of the

R1.2

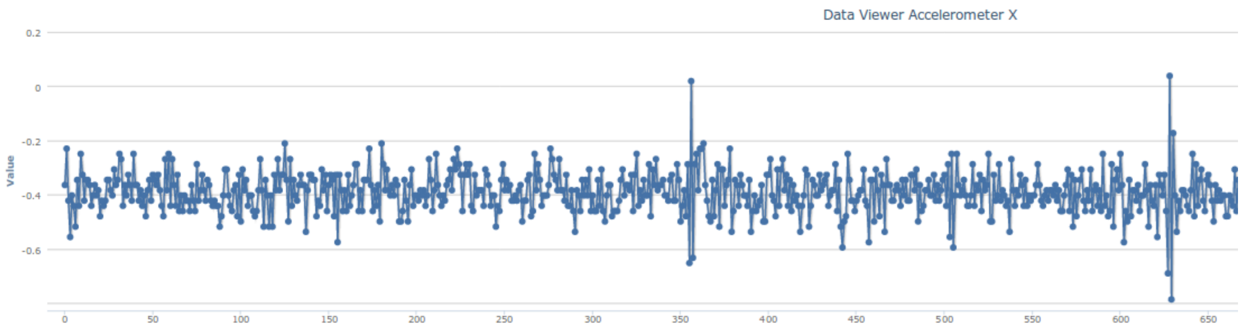


Figure 3.12: X axis of our accelerometer

elevated stand the vibrations weren strong enough to propagate from the keyboard to the table and then to our phone. We then had to test on a thin keyboard that has a direct connection to the table. And for this the apple bluetooth keyboard is exactly the best. After inital testing we found out that the phones accelerometer sampling rate not high enough is. We had an average of 70Hz which for example in [AVT11] they had a minimum of 100Hz and only because of that sampling rate they could successful run the application. If we assume that a person types 40 words a minute. Normalized this means 5 character a word. So speed is approximately 3.333 chars/sec. So if we have at least 100Hz this means we have 30 samples per key press. Less then that we are going to start to loosen some keypresses and then we can't guess the words that have been typed. So our next step was to find a way to raise our sampling rate.

When researching we found out why we had such low sampling rate. One of the reason was battery consumption. Why have a so high sampling rate if less than that would still give a good user experience? And because of that Google has a "Compatibility Definition“ document that comes out for each android version. For getting access to the play store for example, manufacturer have to comply to those rules and meet the minimum requirements. And one of those requirement is that the accelerometer sensor should have a minimum of 50Hz. So thats why manufacturer will meet the bare minimum and not clock higher even if for example the chip has a max sampling rate of Hz as stated in the chips driver source. It's possible that there would be a manufacturer that doesn't block it but none of the devices that we used in this project were capable of using such a high sampling rate.

Now we need to find a work around for our problem. Because our chip

R1.0

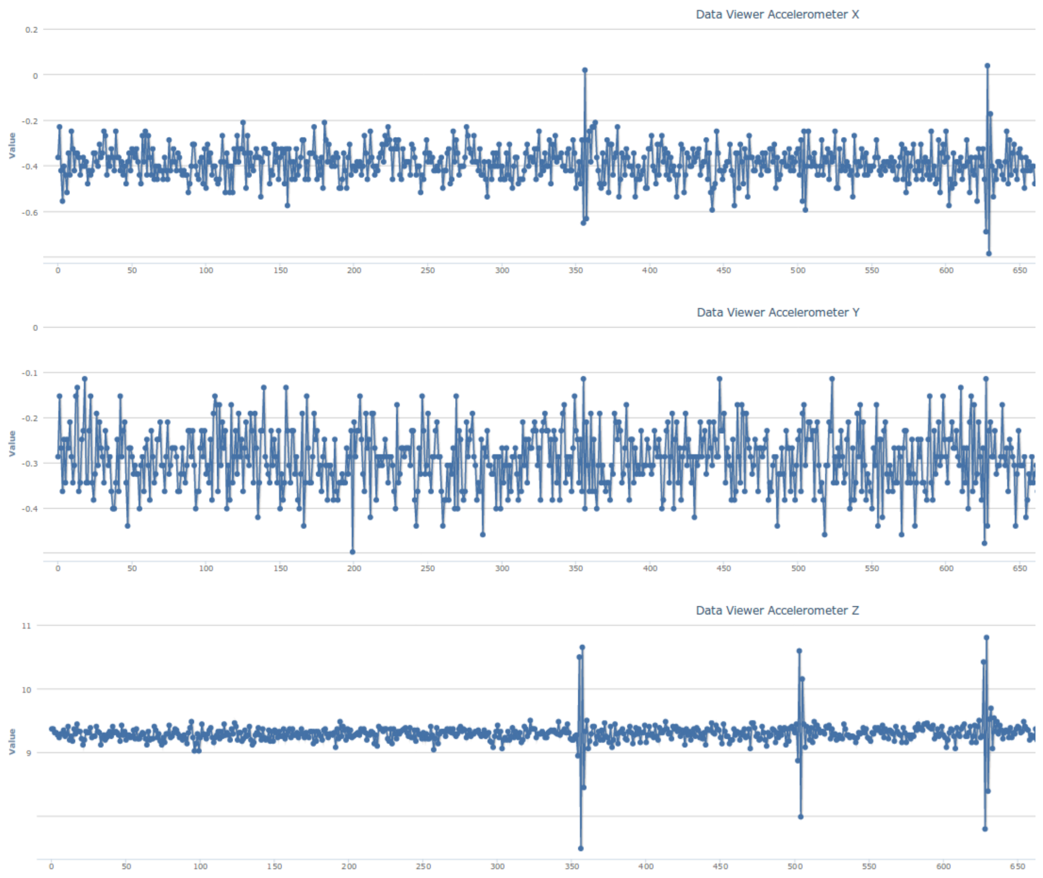


Figure 3.13: 3 Knocks on a wooden table

R1.2

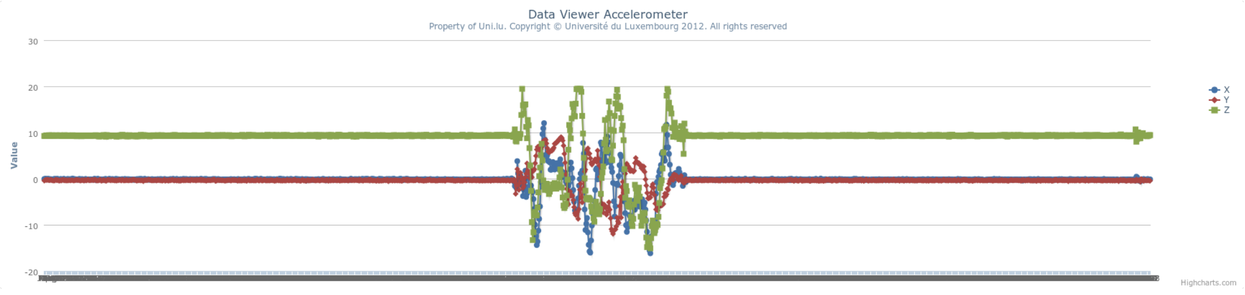


Figure 3.14: 4 Knocks on a wooden table filtered

is more than capable to get higher sampling rate. (KR3DM Accelerometer Sensor max sampling rate at 400Hz which like is as stated in [LZ05] largely enough is) One possible solution would be to avoid the android runtime (figure 3.15). One way to do it provided by Google would be to use the android java native interface.
Also we noticed that the web server solution is a neat and clean solution but when we have too many devices with a high amount of data our browser will start to crash because of low memory. Html5 and javascript are still not ready too handle a such big amount of data. So now we will export the data from our database and then run through octave. It's a open source numerical and graphical tool almost like Matlab.

3.1.7 Phone application in 4.2

Meanwhile a new Android version came out. Unfortunately in the updated docs the sampling rate still was unchanged. So after updating the application and some test run we even saw a decrease of our sampling rate to 60Hz. So this updated version was even more unusable than the one before.

3.2 Environment development with JNI

3.2.1 Introduction

JNI is the Java Native Interface. It defines a way for our android application to directly call native and load dynamic shared libraries (figure 3.16). If doing it right it can be really efficient. One down of is that we have to be careful how we write code because if done wrong it can be inefficient and drain our battery in a matter of minutes or be a memory killer.

3.2.2 Phone app

Our application will mainly be divided in two parts. The native part written in c and then our java part which is going to call our native code. We are going to calculate directly how fast our sampling rate is and then input it on the phone's screen and also on logcat so that there will be no delay nowhere which would slow us down.

The code snippet for the java part. We must pay attention of the name of our library. It has to be well formatted and have the same name as the file otherwise is not going to work.

R1.2

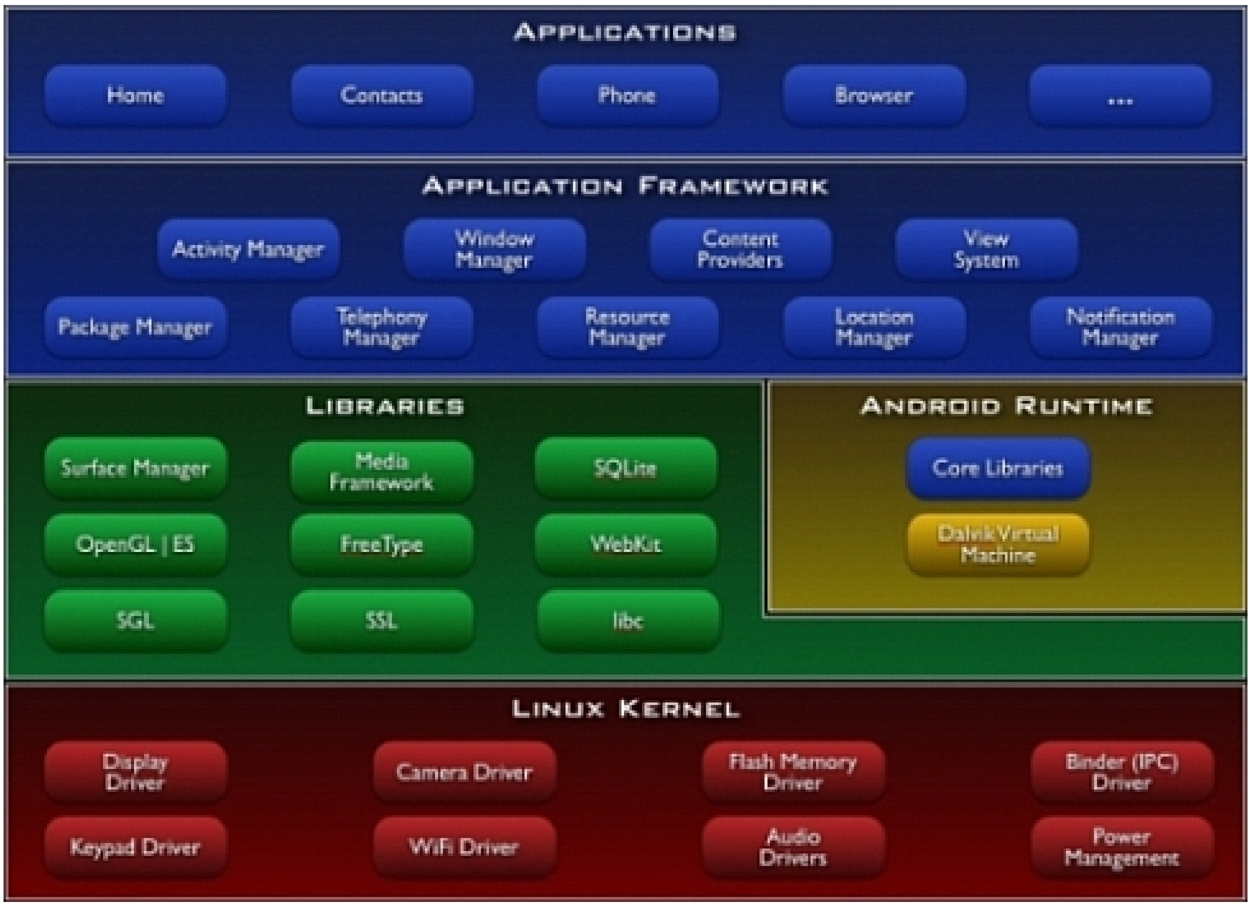


Figure 3.15: Android Structure

R1.2

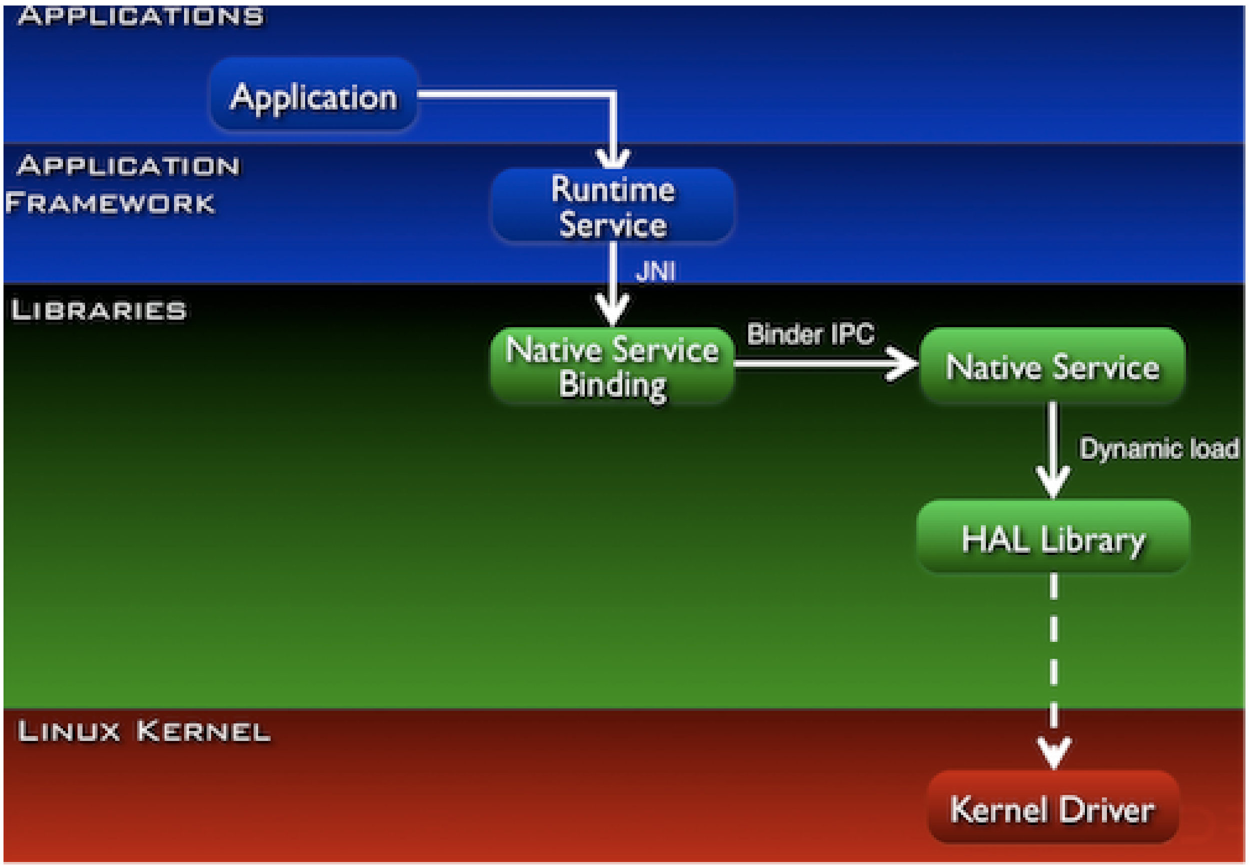


Figure 3.16: Java Native Interface

```
public class Jni extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView  tv = new TextView(this);
        tv.setText( stringFromJNI() );
        sensorValue();
        setContentView(tv);
    }

    public native String  stringFromJNI();
    public native void  sensorValue();
    public native String  unimplementedStringFromJNI();
    static {
        System.loadLibrary("rate");
    }
}
```

For the native code we have first to initalize our sensors and then read. But for now we are going to just calculate the sampling rate to see if we are fast enough. Here the code snippet for initalizing and calculate the duration of a sample collection of 100 samples.

```
    //initalizing our sensors
    accSensor = ASensorManager_getDefaultSensor(
      sensorManager, ASENSOR_TYPE_ACCELEROMETER);
    gyroSensor = ASensorManager_getDefaultSensor(
      sensorManager, ASENSOR_TYPE_GYROSCOPE);
    magSensor = ASensorManager_getDefaultSensor(
      sensorManager, ASENSOR_TYPE_MAGNETIC_FIELD);

    sensorEventQueue = ASensorManager_createEventQueue(sensorManager,
    looper, 3, get_sensor_events, sensor_data);

    ASensorEventQueue_enableSensor(sensorEventQueue, accSensor);
    ASensorEventQueue_enableSensor(sensorEventQueue, gyroSensor);
    ASensorEventQueue_enableSensor(sensorEventQueue, magSensor);

    //reading our sensor
    if(event.type == ASENSOR_TYPE_ACCELEROMETER) {
    //LOGI("accl(x,y,z,t): %f %f %f %lld",
    event.acceleration.x,
    event.acceleration.y,
    event.acceleration.z,
    event.timestamp);
    if(accCounter == 0 || accCounter == 100){
  LOGI("Acc-Time: %lld (%f)",
  event.timestamp,
  ((double)(event.timestamp-lastAccTime))/1000000000.0);
  lastAccTime = event.timestamp;
  accCounter = 0;
}
    accCounter++;
    }
```

We then have to compile the c code separately so that the library can be used from the java application.

3.2.3 Conclusion

This method unfortunately doesn't work either. We have a constant sampling rate of 50Hz. The java native interface is not faster but slower because it's event driven and we don't have control about how fast the events should be triggered.

3.3 Interfacing with the linux kernel

3.3.1 Introduction

With the SDK and JNI is this project not feasible. So one last method that we are going to try is to avoid the entire Android part and go directly to the linux kernel. This method will need root access to the phone and the code will be device specific. So we have to rewrite code for each device. Also we need the sources from the drivers because otherwise it would be a blind work to try to access the sensors directly.

3.3.2 Root access

We first need root access. And because this is a developer phone this is easily done. We need to make a quick update of the phone with the root binary and make the give the him execute permission. After that we install also busybox. Busybox is a small package containing a lot of Unix utilities and combining them into a single executable. Because normally commandy like rm, cp,... are not available. So this way we have all the commands that we will need.

3.3.3 Development ioctl and drivers

Luckily we have the sources for the nexus s's accelerometer. And there we can see that the driver has a maximum sampling rate of 400Hz (figure 3.18). Like stated we will do an ioctl to our accelerometer and read the data directly. Unfortunately it's not that easy. The android system server has all the drivers under lock like if he wanted to use it the all time even when not using. So when we try to make open our accelerometer he will give us an error stating that device is busy. To avoid this problem we simply kill the android system server. Then the phone is restarting and when he's doing we open our accelerometer and take control over it. Meanwhile the phone has finished booting and everything still works with exception that the phone it's not going to rotate because he couldn't take control over the accelerometer because we had it. Now we have access to the accelerometer.
For compiling c code and then run it on the phone we have to install the gcc arm libraries for successfully compile it and then run on the phone natively. After successful compiling we can then run our code on the phone. Because this code is running natively and prioritary this is a really ressource hungry and battery exhausting way.

```
    //here the command for compiling
     arm-linux-gnueabi-gcc -static -march=armv7 code.c -o output

     //c code
/* kr3dm ioctl command label */
#define KR3DM_IOCTL_BASE 'a'
#define KR3DM_IOCTL_SET_DELAY       _IOW(KR3DM_IOCTL_BASE, 0, int64_t)
#define KR3DM_IOCTL_GET_DELAY       _IOR(KR3DM_IOCTL_BASE, 1, int64_t)
#define KR3DM_IOCTL_READ_ACCEL_XYZ  _IOR(KR3DM_IOCTL_BASE, 8, \
                                            struct kr3dm_acceldata)
        if ((fd = open(argv[1] , O_RDONLY | O_NONBLOCK)) == -1) {
            perror("open");
            return 1;
        }

if (ioctl(fd ,KR3DM_IOCTL_SET_DELAY,&delay)==-1){
   printf("ret val %d\n%s\n",ret,strerror( errno ));
return 1;
}
time_t start = time(NULL)*1000;
time_t end=start+1;
while(((end-start)/1000)<=60){
  if (ioctl(fd ,KR3DM_IOCTL_READ_ACCEL_XYZ,&sum)==-1){
  printf("ret val %d\n%s\n",ret,strerror( errno ),&sum);
  return 1;
  }
  count=count+1;
  end = time(NULL)*1000;

    }
printf("Time: %d->Count: %i\n",((end-start)/1000), count);
fclose(p);
close(fd);
    return 0;
```

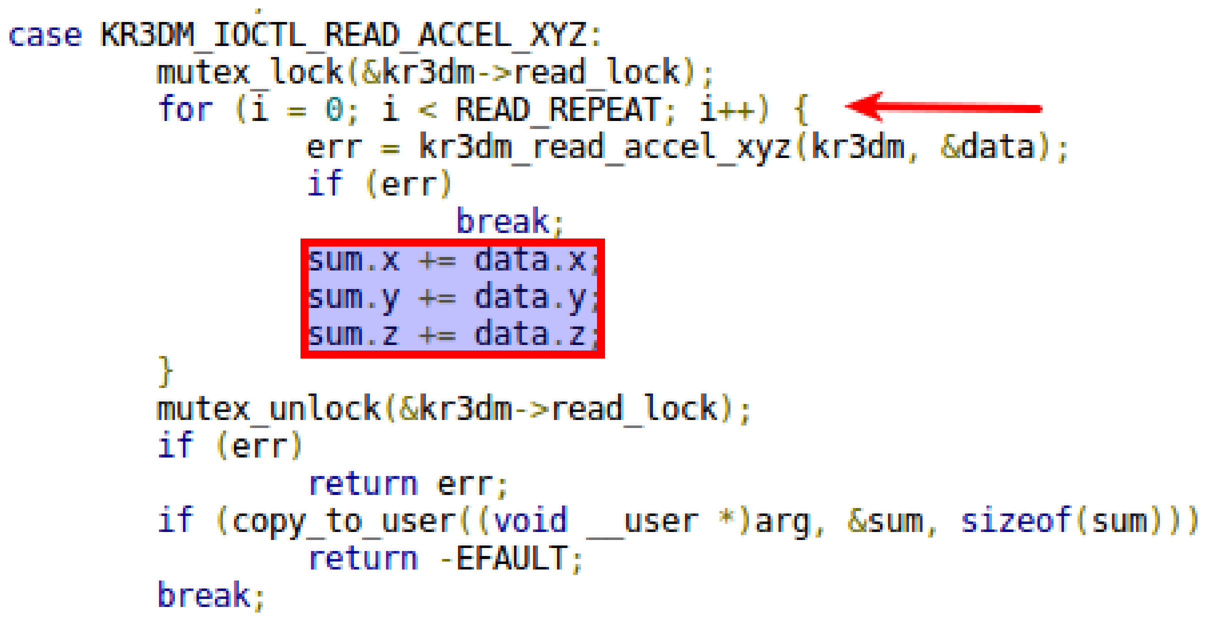
3.3.4 Conclusion

We could successfully run our code and read the accelerometer natively and avoid the entire android runtime. But unfortunately we could achieve the desired sampling rate. We only could get 50Hz max. In figure 3.17 we see why. He summing up to a total of 8 samples and then he outputs it. So if divide 400 with 8 we get exactly 50Hz which is what we are getting out. So like prevous attempt this one also didn't work.

3.3.5 Future possible solutions

One possible solution that one could try is to do use i2c to attempt to read the accelerometer and avoid ioctl. Because we have the source code of this chip this could be a viable solution.

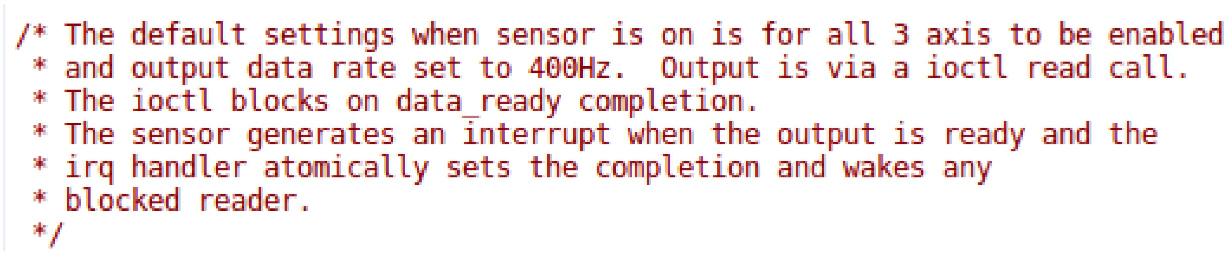
R1.2



```
case KR3DM_IOCTL_READ_ACCEL_XYZ:
        mutex_lock(&kr3dm->read_lock);
        for (i = 0; i < READ_REPEAT; i++) {
                err = kr3dm_read_accel_xyz(kr3dm, &data);
                if (err)
                        break;
                sum.x += data.x;
                sum.y += data.y;
                sum.z += data.z;
        }
        mutex_unlock(&kr3dm->read_lock);
        if (err)
                return err;
        if (copy_to_user((void __user *)arg, &sum, sizeof(sum)))
                return -EFAULT;
        break;
```

Figure 3.17: Kr3dm accelerometer ioctl problem

R1.2



```
/* The default settings when sensor is on is for all 3 axis to be enabled
 * and output data rate set to 400Hz.  Output is via a ioctl read call.
 * The ioctl blocks on data_ready completion.
 * The sensor generates an interrupt when the output is ready and the
 * irq handler atomically sets the completion and wakes any
 * blocked reader.
 */
```

Figure 3.18: Source code from Kr3dm accelerometer

Chapter 4

Conclusion

We have tried many attempts. SDK, JNI, Kernel direct access. None of these worked with the hardware that we had in hand. So we can say that the vulnerability that the iphone [AVT11] has can not be applied to the android plattform. It's possible that somewhere some phone has the driver implemented in such way that it has a high sampling rate. High enough for this project. But for now from three different manufacturer for our four different devices reading at such high speed is not possible and because of this, this project is not feasible.

List of Figures

Bibliography

[Agr04] Dmitri Asonov Rakesh Agrawal, *Keyboard acoustic emanations*, no. 1, 9.

[AS12] Highsoft Solutions AS, *Free to use javascript charting library*, 2012.

[AVT11] Henry Carter Arunabh Verma and Patrick Traynor, *(sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers*, no. 1, 12.

[Geh11] Jonas Gehring, *Open source graphing library for android*, 2011.

[LZ05] J. D. Tygar Li Zhuang, Feng Zhou, *Keyboard acoustic emanations revisited*, no. 1, 28.

[Sis07] Gurú Sistemas, *Open source data grid php library*, 2007.