Fachhochschule
Münster University of
Applied Sciences

Fachbereich Elektrotechnik und Informatik

Bachelor´s Thesis

Human Activity Recognition based on an application for the Android operating system

Jens Ostrowski

September 16th, 2011

Advisors: First Examiner: Prof. Dr.-Ing. Peter Glösekötter
Second Examiner: Prof. Dr.-Ing. Ignacio Rojas Ruiz

Abstract

We present an application for a smartphone that helps to recognize the daily living physical activity of the user only by using the available sensors in the device. The Android operating system serves as platform for the application.

The application is going to work in a “passive” way, meaning, once activated, it runs in the background. It collects data from the sensors (at first, only acceleration data is obtained) and sends them for an analysis to a web server. Via an interface the user can receive access to the data.

One of the most remarkable advantages of the system is the non obtrusiveness. The user is in no way limited in his movement, because the application does not require the device to be attached to a specific part of the body. The user uses the Smartphone in the way he always does. No other devices are required.

The application can collect lots of data from users all over the world. These data can be uploaded to the laboratory web server to be analyzed in the future. With the help of this global monitoring system we can find out more about the physical activity of the users and increase the accuracy of the application.

The technical features of the application will be a simple and understandable interface every consumer can easily use. The application will use the integrated sensors of the smartphone, while we try to limit the consumption of energy.

Index

1. Introduction

In our modern society the challenges to modern medicine are enormous. And as the average age will increase more and more in the next decades, even more challenges are about to come.

The percentage of people in the EU of age 65 and older is predicted to increase from 17.1% in 2008 to 30% in 2060, while the number of people of age 80 and older is expected to almost triple from 4.4% to 12.1% in that period [Eur01]. As a result of this, calculations assume that health care spending will increase between 1% and 2% of the gross domestic product (GDP). On average, this would increase approximately an amount of 25% in spending on health care, as a share of GDP, in the next 50 years [Eur02]. Therefore, adding technology effectively to health care systems could help to decrease public spending on this topic. One of these emerging technologies is the human physical activity recognition.

The recognition of human physical activities is applicable in various fields, for example chronic disease management, rehabilitation systems, disease prevention and as indicator to the personal health status. One of the major subjects on the health sector is monitoring the physical activity of elderly. For example are falls one of the major risks for their safety and a huge obstacle for their independence. This risk is even increased when some kind of degenerative disease affects them.

To this day, several studies have achieved successfully recognition of the most relevant activities like walking, running, sitting and standing still. However, most of the results were received under laboratory environments, with a direct supervision by the researcher. Therefore, those outcomes can hardly be transferred into real-life situations.

Another strategy, as persuade by some studies, is to try to apply a semi-naturalistic approach. This approach tries to get more realistic results by finding a balance between laboratory and natural monitoring. The actual monitoring is concealed from the subject so the activity is performed in a more natural state. Thus, this data is closer to the natural behavior of the subject because of its unawareness.

A naturalistic way of monitoring the physical activity would be ideal. The monitored person is not aware at all of being monitored. He/she behaves in completely normal patterns and the researcher has no influence on the subject.

This project is an attempt to monitor the physical activities of persons in the most naturalistic way possible. With the help of the smartphone´s accelerometers the movement of the user can be monitored in his natural environment. The application runs fully independent of the normal use in the background, the user is not influenced by it. In this way, he/she is not aware of being monitored and can act completely normal. The received data can be uploaded to a web server via a wireless internet connection, where it can be analyzed and used for further research. By using the popular Android platform, a huge number of subjects can be reached and the accuracy can be increased by comparing several different subjects performing the same activity.

The rest of the paper is organized as followed. The second chapter compares and examines currently available smartphones based on the needs of the application, and analyses the state of the art in human physical activity recognition by describing different applications available. In the third chapter the Android operating system as a platform for this application is introduced and described. The fourth and fifth chapters describe the implementation of the application, where the fourth chapter describes the application and every part of it in general and the implementation in detail is described in the fifth chapter. The sixth chapter offers a conclusion of the given task and looks for the prospects of the future.

2. Analysis of available hard and software

2.1 Introduction

The development of smartphones, which are high-end mobile phones with more computer functionalities and connectivity than usual cell phones, is still in the beginning. Today´s models are not only used for phoning and messaging, they also serve as portable media player, web browser or GPS navigation system. The consumers have increasingly higher demands on the technique, therefore both hardware and software components will develop rapidly the next years. However, because of this relatively young market new ideas and innovations are very likely and expected frequently.

One of these ideas is the Human Activity Recognition Application. However, before this application and its functionalities are described, a short overview over the available hardware and software is given.

At first, this chapter describes the current situation on the smartphone market. Different types of smartphones are introduced and compared to the needs of the Human Activity Recognition Application.

Second, already available software from the Android Market is examined. As examples, three different applications are examined and compared to the application introduced here.

2.2 Hardware and smartphones

An important task for the Human Activity Recognition Application is the hardware. As operating system Android is chosen, which means that only smartphones running Android need to be considered.

The Android operating system (Android OS) is a product of the Open Handset Alliance, a consortium of hard-, software and cell phone companies lead by Google. It was founded in November 2007 probably as a reaction to the commercial success of Apples iPhone. The formulated aim was to develop and apply open standards for devices and programs [Ope01].

As a result, the Android OS, mainly developed by Google and based on the Linux kernel, was published. The source code is provided open source under an Apache license [And01]. The development of Android OS started a few years before the consortium was founded. The first Android smartphone was the HTC Dream, published in October 2008 exclusively by T-Mobile.

Due to the fact that the Android OS, as opposed to Apples´ iOS, runs on many different smartphones produced by several companies, a closer look at the different phones is necessary. As the Open Handset Alliance includes many of the words largest producers of mobile devices, such as Samsung, Sony Ericsson, HTC or LG, the number of available Smartphones has increased enormously in the last 4 years. And in the next years the number is projected to increase even more. Already, the smartphone market is quite complex.

For this comparison all smartphones available on the market, but also older ones are examined. The focus will be components of directly influence to the Human Activity Recognition Application. The three most interesting parts are the processor architecture and speed, the version of Android OS, and of course the integrated sensors.

When it comes to the processor architecture, the most important component is not the processor´s speed but the available memory. If the operating system discovers a lack of memory it shuts down the unimportant applications, i.e. the ones that do not run in the foreground. As the Human Activity Recognition Application is such an application running in the background, it is vulnerable to be shut down if the system runs out of memory.

The version of Android OS is in this respect not that important, because most producers offer an update for their phones if newer versions are available and all programs written in Android are upwards compatible. The Human Activity Recognition Application is developed for Android 1.5, so it will run also in every version above 1.5.

The third aspect, the sensor equipment, is both, important and unimportant. Of course accelerometers are indispensable for this application, but almost every Smartphone on the market is equipped with 3-axes accelerometers. The newest high-class Smartphones even have 6-axes sensors.

For a better overview, the available Smartphones are divided into three groups. In the first group are the latest Smartphones, appeared in 2011. Their features are dual core processors, Android 2.2 Froyo or 2.3 Gingerbread. The integrated sensors are, among others, accelerometer and six-axis motion sensors. Examples are the LG Optimus Speed P990, the Samsung Galaxy S2 or the HTC Sensation.

In the second group are high class Smartphones from the late 2010 and early 2011, e.g. the Samsung Galaxy S, Sony Ericssons Xperia arc or the HTC Desire. These devices work with a single core processor and a lower clock speed than models from group one. In addition, they mostly have less memory. The Android version is in most cases 2.2 Froyo or, in newer models, e.g. Sony Ericsson Xperia arc, 2.3 Gingerbread. In comparison with smartphones from the first group, these models are equipped with fewer sensors, e.g. only three axes motion sensors and no gyroscope.

In the third group are beginner and older smartphones from 2010 and 2011. These smartphones have partially huge differences among each other, but are all characterized by their lack of features compared to the second group. Most of them work with a single core processor and clock speed around 600MHz. More critical is that some models do not have the sensors important for this project. Here, for examples are the Acer Stream, the Sony Ericsson Xperia mini or the T-Mobile G1 (HTC Hero). A short overview can be seen in Image 2.1.

				Group 1		Group 2		Group 3		
				Samsung Galaxy S2 i9100	LG P990 Optimus Speed	Sony Ericsson Xperia Arc	Samsung Galaxy S i9000	Acer Stream	Sony Ericson Xperia X10 mini	T-Mobile G1
				1. Quater 2011	1. Quater 2011	1. Quater 2011	2. Quater 2010	3. Quarter 2010	2. Quater 2010	1. Quater 2008 (first Andriod smartphone)
Operating System version				Android 2.3 Gingerbread	Android 2.2 Froyo	Android 2.3 Gingerbread	Android 2.2.1 Froyo	Android 2.1 Eclair	Andrion 2.1 Eclair	Android 1.1
Processor				Dual-Core Samsung Exynos 4210 (1.2 GHz)	Dual-Core ARM Cortex A9 (1Ghz)	Single-Core Qualcomm Snapdragon QSD8255 (1GHz)	Single-Core Hummingbird (1GHz)	Single-Core Qualcomm QSD8250 (1GHz)	Single-Core Qualcomm MSM7227 (600MHz)	Single-core Qualcom MSM7201A (528MHz)
Memory				1 Gbyte RAM	512 MB RAM	320 MB RAM	512 MB RAM		512 MB RAM	192MB RAM
Sensors	motion sensor			6-Axis	6-Axis	3-Axis	3-Axis	3-Axis	3-Axis	yes
		accelerometer		yes	yes	yes	yes	no	no	no
		gyroscope		yes	yes	no	no	no	no	no
	approach sensor			yes	yes	yes	yes	no	yes	no
	brightness sensor			yes	yes	yes	yes	no	yes	no
	fall sensor			no	no	no	no	no	no	no
Energy	battery			Li-Ion	Li-Ion	Li.Ion	Li-Ion	Li-Ion	Li-Ion	LI-Ion
	Power			1630mAh	1500 mAh	1500 mAh	1500 mAh	1400mAh	1200 mAh	1150 mAh
Interface	USB-Version			2.0	2.0	2.0	2.0	2.0	2.0	1.0(?)
	Bluetooth-Version			3.0	yes	yes	3.0	2.1	2.0	2.0
GPS										
Internet	HTML									
		xHTML		yes	yes	yes	yes	yes	yes	yes
		cHTML		yes	yes	yes	yes	yes	yes	yes
	Browser			Webkit	Webkit	Webkit	Webkit	Webkit	Webkit	Webkit
	Java			MIDP 2.1	MIDP 2.1	MIDP 2.1	MIDP 2.1	MIDP 2.1	MIDP 2.1	MIDP 2.1

Image 2.1: Comparison of different smartphones

In summary, almost every smartphone available on the market at the moment is equipped enough for the needs of the Human Activity Recognition Application. Only a very few older models are not able to run this application.

Differences in the equipment between high class models from the first group and beginner models from the third group will affect the applications speed and usability, when for example loading time is increased.

2.3 Market analysis

At the moment, the Android market offers more than 480,000 applications available for download, and the platform is extended with more than 30,000 new apps every month [Lib01]. A large percentage of the offered applications are games, but also the fitness and health sector, to which the here introduced app is classed among, is highly represented.

Nevertheless, only very few applications attend the topic of human physical activity recognition. For the most part, the applications support workout and exercising by counting calories, planning exercise routes or issue exercise plans. But some interesting applications can be found.

One example is the application “Mover”, developed by Fraunhofer Portugal Research Center for Assistive Information and Communication Solutions (Fraunhofer AICOS) [Fra01]. The aim of this app is to detect the movement of the user and categorize him/her into several different activity levels. That takes place by reading signals from the accelerometers and summing them up. By calculating the average movement throughout the day, the user is categorized into seven different levels of activity, from sleeper to hyper (active). Additionally, a fall detection algorithm is added, which is combined with emergency call in case the application detects no movement after the fall. Summarizing, this app does not actually detect the quality of activities but only their quantity. However, it is still under construction.

Another example is “Sensor Logger” [Sen01]. This application tries to detect the activity performed by the user over a short period of time. When started, the application obtains data for about one minute and displays the calculated activity. It uses the accelerometers, the magnetic field sensors, and the orientation sensors, and tries to identify walking (normally, up stairs, and down stairs), standing, sitting, travelling by bus, travelling by car, and dancing. After the detected activity is displayed, the user has the chance to correct the activity, if the application detected it wrong. After that, the data is send to a webpage, where the user can display a graph of his activities.

In a test, the application was wrong in over 80% of the cases. Not even standing still was detected correctly every time. Walking was detected in almost every case, but if the person went up stairs, down stairs or normally was incorrectly detected almost every time.

The developer of the “Sensor Logger” application further offers the “Activity Recorder”-App. This app performs activity classification similar to the “Sensor Logger”, but does this periodically in a background service and stores the detected activities in a list. However, the results are not very good, the detected activity is almost never the one performed, fast changes in activities cannot be detected at all and even when the phone is not moved at all, the results vary.

Both applications detect many of the user´s movement incorrectly. As the “Sensor Logger”-App had a few rights, the “Activity Recorder”-App had almost no correct result. But both are still under construction and updated regularly.

These three examples are only a small overview, but represent the present state at this topic.

Generally, the human activity recognition based on applications for Android is in the early stages of development. There are only a few available programs and all of them suffer from various problems as described above. However, first steps are made and new developments will come up in the next years.

3. Android operating system

3.1 Introduction

When Apple launched its iPhone in 2007, a new aspect of the smartphone market came into the public focus: The App-Store, an Apple operated and moderated marketplace for software from third-party-suppliers, allowed the user of Apple-mobile devices various modification of his device by downloading additional software.

Together with the App-Store, Apple published the iPhone Software Development Kit (SDK), which allowed an application development directly to the special hardware of the iPhone. The idea of an Application Programming Interface (API) explicitly designed for mobile devices was not new. Although the Java Micro Edition was available on many mobile devices before, the App-Store allowed an easy access to the software, even from mobile devices, making the iPhone Apps much more popular than Java Apps. The increasing success of the iPhone, in 2009 Apple sold almost 25 million units [Gar01], was another reason why developing software for mobile devices became more and more influential to software development in general.

Together with the operating system Android, the Open Handset Alliance published the Android Software Development Kit (SDK), which allowed the development of applications for the system on the basis of an extensive API. It is based on a downgraded version of the Java standard.

As well as Apple, Google offers a market place for applications, where registered developers can upload their programs. The Android Market Place offers an enormous number of free applications and applications to pay for.

Since its launch in 2007, the Android operating system has been incredibly successful. In the first quarter of 2011, it became the most sold operating system on smartphones worldwide [Gar02]. Experts believe that within the end of 2012, nearly half of the smartphones worldwide will run Android as operating system [Gar03].

This chapter describes the structure of Android OS. All functionalities used in this project are described in detail in the following paragraphs.

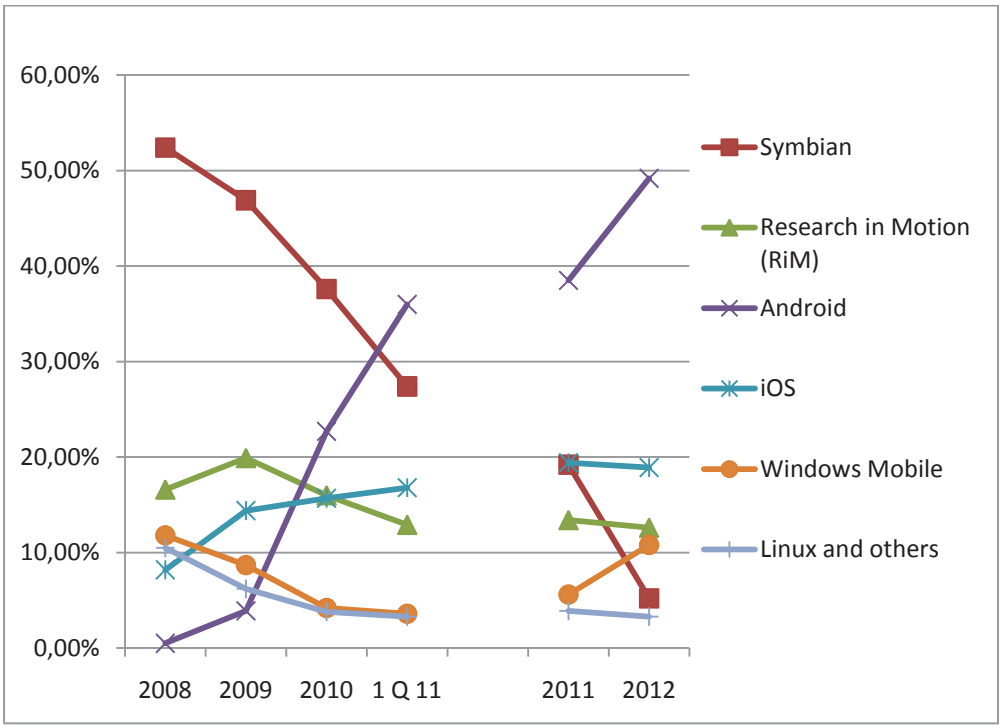

Image 3.1: Development of operating systems 2008 – 1st quarter 2011 and future; projected values for 2011 and 2012 on the right site

3.2 Basics of the operating system

The structure of Android can be divided into four layers: kernel, libraries, application framework and applications. The lowest layer of the architecture is a 2.6 Linux kernel, which is responsible for driver, user and energy management. Like other operating systems, this is the connection to the hardware.

The layer above includes a set of libraries written in C/C++. They are exposed to developers through the application framework and include some core libraries like WebKit, OpenGL, SQLite and others. The complete list can be found in [And02].

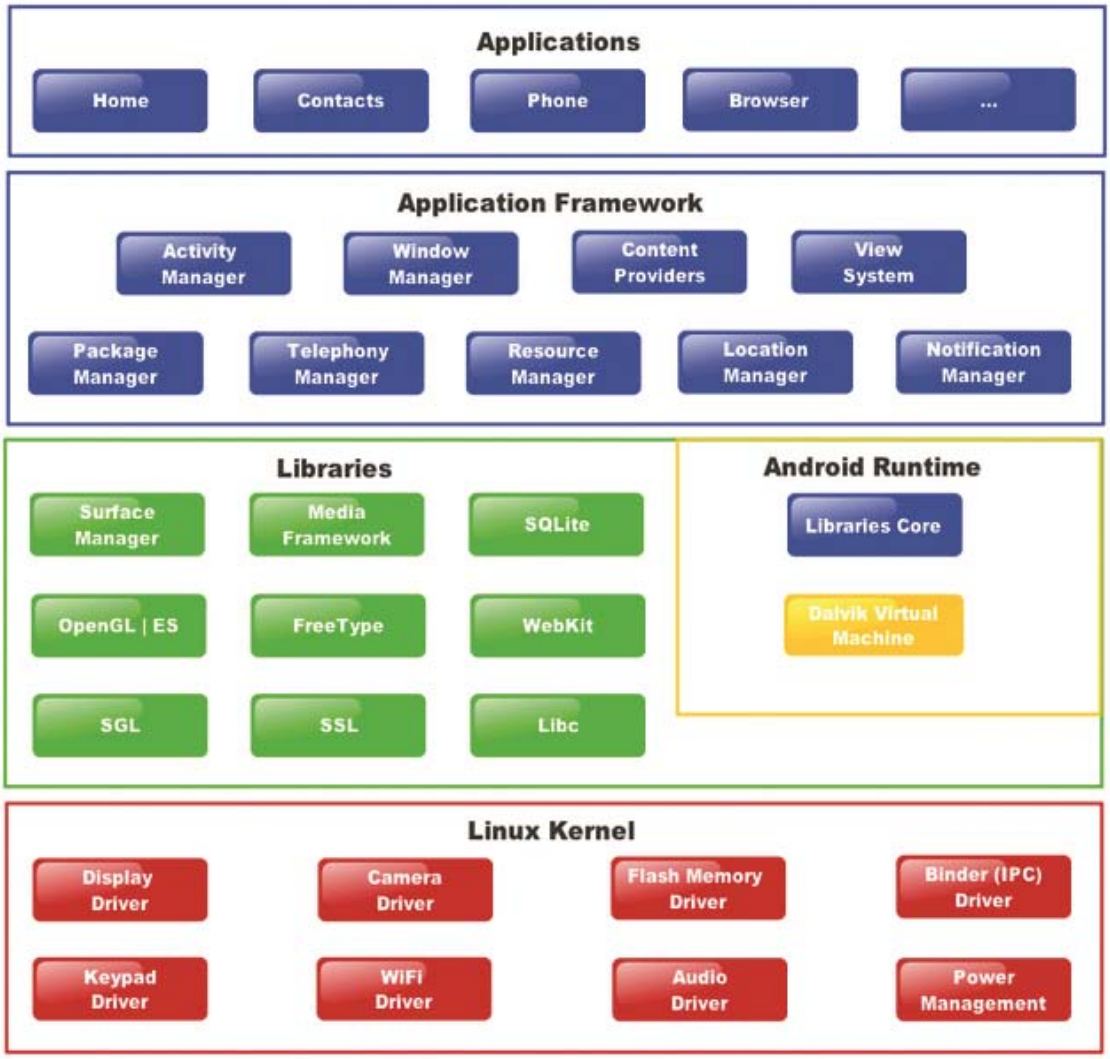

Image 3.2: Android system architecture [And02], compiled by Alvaro Fuentes Vasquez for Wikipedia [Fue01].

The link between the hardware near components, resources and libraries and the actual application is the application framework. This is the main part of the Android Software Development Kit (SDK). It consists of predefined objects for displaying data as well as classes for managing data, using built-in hardware functions and the transact of the application lifecycle. Further, it allows full access to the same framework APIs used by the core applications. In the following parts a lot of them will be examined more precisely.

The topmost level of the Android architecture is the application itself. These are the core applications like email client, SMS program, browser, and programs developed by third-party suppliers. All applications are written using the Java programming language. The Android SDK Tools compiles the code

into an Android package file (.apk) and installs the application on the mobile device.

3.3 Process management and virtual machine

For safety and performance reasons, Android tries to run every application as discreet and encapsulated as possible. Therefore, it uses mainly three different procedures:

Each application appears as a different user on the kernel layer to the operating system. When installed, the system already gives permissions in a way that only the application itself has access to its own files and data. But by giving special permissions, it is possible to give one application access to another, if they are from the same developer.

Additionally, every program runs in its own Linux process. In this way, Android is able to shut down a process when it is no longer needed or when resources are needed for other applications.

According to usual practice for Java programs, Android uses a virtual machine (VM) to run its programs. Android starts its own virtual machine for every application. That way, the executed code stays isolated from other applications. Similar to the user IDs on the kernel layer, Android allows packages with the same signature to run in the same thread together, which means, they use the same virtual machine.

Android does not use the standard virtual machine used by the Java Runtime Environment. It uses a Dalvik virtual machine which is adapted to handle the fewer resources given by a mobile device.

3.4 Android Software Development Kit

The Android SDK contains over 1500 (2011) classes by various creators. Nearly half of them originate from the Java Development Kit (JDK) by Oracle/Sun, including most of the standard functionalities. These include primitive data types and further threading and streams and also the collection framework, bridge classes for cryptology and compression algorithms, XML and OpenGL-ES. Not included are graphical components like AWT and Swing.

The biggest part of the API, besides the JDK classes, consists of Android specific classes in the Android package. These contain basic classes of the

Android components, wrapper implementations for using hardware components, classes to access the standard libraries and basic implementations for graphical user interfaces.
Beside the class libraries, the development kit offers a lot of useful tools. First of all, there is the emulator, with which the written code can be tested on the development computer. Others are the Dalvik compiler, a packaging tool to create .apk files and a debugger to test an application during its runtime.

3.5 Structure of an Android program

The structural approach of Android is to create a component based application. That means that the applications are no longer considered as self contained functional units, but the possibility is given to share functionalities and features with other applications. Thus, a new implementation of heavily used functionalities can be avoided. Consider the following example. An application for exchanging massages does not need its own contact database. The component based approach of Android allows the developer to ask the system for an existing contact database and get access to it.
Android does not divide the program and places parts of the application at disposal. The developer can decide himself, if and to which extend he wants to allow access to the functionalities of his application. The mechanism to place functionalities of an application at disposal is called intent and will be discussed later.
Let us take a closer look at the Android manifest first. That is a XML document, which describes the structure of the program, the available components and the application-specific requirements. The file itself has to be named AndroidManifest.xml and is structured as shown in Listing 3.1.

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.strowski.accelapp"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <application android:icon="@drawable/icon" andro-
id:label="@string/app_name">
        <activity android:name=".accelMain"
```

```
                    android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category andro-
id:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Listing 3.1: The Android manifest file

Every data of the manifest is placed within the manifest-Tags. The `xmlns:android` attribute is essential and given by default. The internal version number of the application is given by `android:versionCode`, the public version name by `android:versionName`. The attribute package shows the name of the package which contains the application. The same conventions as in normal Java applications are recommended.

All components containing the application are defined inside the application element. These components can be services, activities, content provider and broadcast receiver. They will be discussed later.

Among others, the application element serves to give necessary permissions to specific contents. In the `android:label` attribute the name of the application is determined. This name is shown in the list of apps on the mobile device, for example. Also, an icon for the app can be specified under `android:icon`. Other settings are optional, for example the runtime environment can be influenced.

The following elements define the settings and requirements of the application. At first, the Android SDK version the application resorts to has to be specified. This happens in the `uses-sdk` tag. By doing so, the system checks if all requirements of the application are available.

With the elements `uses-configuration` and `uses-feature` other specified requirements can be given to the hard- and software. These include possible input units like the touchscreen or the trackball, or specified hardware add-ons like flashlight.

3.5.1 Permissions

A very important part of the manifest are the permissions. In this way the developer declares that the program wants to use certain basic functionalities of the device. The element `uses-permissions` is used for this, to which the functionality is delivered as an attribute. There are more than 100 different permissions given by Android concerning the access to almost every component and standard data of the device, for example access to the internet, Bluetooth, contact databases or energy management.

Installing an application from the Android Market on his device, the user is informed which permissions the program claims. Based on that, the user can choose to install the application. A selective ban of certain permissions does not exist. The option to withdraw permissions from installed applications is not provided.

If a developer wants to use a protected function like the access to the positioning system of the device, although he did not request for the permission, the system reacts with a security exception and stops the use of the corresponding feature. If the developer does not catch that exception, the program stops.

Because Android is content based, the developer also has the option to protect parts of his program with permissions. They are also declared by permission elements in the manifest file and get a unique name by the attribute `android:name`. As it is common in Java language, the name should be unique, because permissions are application comprehensive and otherwise it could cause conflicts in the system.

3.6 Activities

All types of components in correspondence with the user, namely components with a graphical interface, are of the type activity. Every application defines a specific activity as a starting point, which is shown on screen when the program starts.

Every activity a program wants to use has to be declared in the manifest. This is done inside the activity tag, which is inside the application element. An activity which is not declared cannot be called and the virtual machine reacts with an exception.

The most important part of the activity tag is the attribute `android:name`. It contains the class name of the activity. The full name has to be declared, including the package. To avoid that, a relative path can be declared by starting the name with a dot. In this way, the name referring to the package is given in the manifest.
Further attributes are optional. Amongst others, the alignment on the screen or the name of the activity window can be adjusted.
There are different ways to start an activity. On the one hand, one specific activity is called when the application is started, on the other hand, it is possible to start an activity from an existing activity. Here it is important to distinguish if the activity is supposed to run discretely or if it should return any result. Depending on that, the activity can be called by `startActivity(Intent)` or `startActivity(Intent, int)`. Neither method provides a return value. A closer look into the process of calling activities will be in the part of the intents.
If the activity is supposed to return a result, this result has to be saved. The method `setResult(int)` is responsible for that. There for standard constants are available. For the return of additional information, an intent with data can be given to the method `setResult(int, Intent)`.
When an activity is started, Android saves the related data on a stack. Also, all previous activities started since the last call are saved on this stack, with the most recent activity on top. To get from the current screen back to the one used before, the user needs to press the back button of the device. This feature is very important, if an application is made of various user interfaces calling each other or sending data among each other. In this way, a return from a sub interface to the main interface is ensured without losing data.
Every called activity automatically starts a dedicated life cycle, which passes through several stages, depending on the state of the activity. In general, this life cycle depends on whether the activity runs in the front, background or is half visible. Only activities that are not covered by others are really active. All activities in the background are in an inactive state and can be shut down by the system if the resources are needed otherwise.

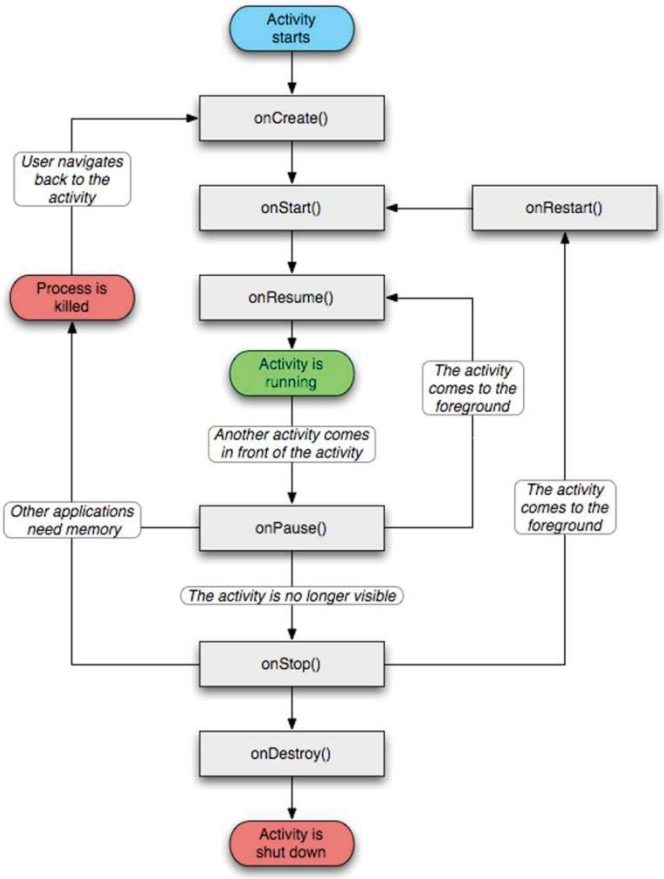

Image 3.3: Activity lifecycle [And03]

Every life cycle starts with the method `onCreate(Bundle savedInstanceState)`.This method is called every time an activity is created for the first time. The fundamental operations necessary for the first display are done here. This includes creating the users interface and loading relevant data. As for all standard methods of the life cycle, it is stringently required to call the `onCreate()`-method of the parent class within the `onCreate ()`-method of the own activity by calling `super.onCreate()`. Otherwise, the system throws an exception and stops the program.

The object of the type `Bundle` given to the `onCreate ()`-method is a collection of data representing a status the activity had in an earlier life cycle. In this way a canceled activity can be relegated to the state it was during an earlier run. The method `onSavedInstanceState(Bundle outState)` serves to restore that state. This method is called when a new activity covers the old one. However, this happens only in cases the system expects to jump back to that specific instance of the activity and the activity is shut down by

the system while inactive. If an activity is left by the back button, it disappears from the stack and gets destroyed by the system. In this case, if the user wants to return to that activity, the state has to be saved.

The next steps after calling the `onCreate()`-method are the `onStart()`- and `onResume ()`-methods. Data and resources, which are often used or needed quickly after a short time of inactiveness, should be allocated here. The `onResume ()`-method is the only method guaranteed to be called after a time of inactiveness. However, it is also very important to ensure a high performance in this stage, because a status where the activity is in pause can occur often.

Similar to the generation of an activity, a chain of methods runs while an activity becomes inactive. Contrary to the procedure while activating, not all inactivation methods are guaranteed to be called. At any point the system has the opportunity to shut down the process belonging to the activity. This is necessary when resources are short. In this case, the outstanding methods are not called anymore.

The opposite of the `onResume()`–method is the `onPause()`–method. It is called when the activity does not run in the front screen anymore. Together with the `onResume()`–method it is the only method that will surely run when the activity is paused. That is why saving data at this point is important. Also, resource spending parts should be stopped here. It is important that the activity that overwrites the paused activity runs first, when the `onPaused()`–method is completed.

If an activity is on pause, there are three possible ways to resume it. First, the process can be finished, and then the activity is destroyed. Second, the activity can return to the front, and the `onResume()`–method is called. If the activity is completely covered by another one, the method `onStop()` is called. The time this interim state lasts is not defined. Again, the activity can be destroyed or come back to the front. In this case, the method `onRestart()` is called to relay resources that have been used in the lifecycle before. Then the method `onStart()` is called again.

Alternatively, the activity can be destroyed by the method `onDestroy()`. Here all recourses should be released that have been used during the runtime. When the activity is called the next time, they will be created again.

3.7 Services

The second application components are services. Contrary to activities they do not have a users interface. They run in the background and perform for long running tasks. Usually a service runs in the same thread as the activity. That leads to the danger of getting shut down by the system if the service needs too many resources.

Like activities, services are components to be accessed from outside. They have to be declared in the manifest as well. Appropriately, the service element is designed similarly. The class name has to be committed, furthermore permissions can influence the access from third-parties.

As we said before, a service is designated to run in the process of the application that declared it. This can be influenced in the manifest by the attribute `android:process`. Is a name declared here, the service is run as its own process to which other applications can get access if they have the right permission. Nevertheless, if the name starts with a colon, only the starting application has access to the service.

The life cycle of a service is much simpler than the one of an activity. There are mainly two important controlling methods, `onStartCommand()` and `onStop()`, called for starting and ending the service. The missing visibility makes it redundant to create intermediate results.

Within an activity a service is started by calling the method `startService(Intent)`, where the name of the service is committed with the intent. As a consequence, equal to activities, the starting instance has no reference to the instance of the newly started service. To contact the service, an explicit connection to it is necessary.

The connection to a service, and thus the data exchange, is controlled by binder-objects. These binder-objects represent the connection to the service. To get a binder the method `bindService(Intent, ServiceConnection, int)` is called. The intend identifies the service, with the `int` attribute the assumption in which state the service is supposed to run is given. Here it can be specified, if the service is explicitly started newly or if it should only run as long as the life cycle of the application runs.

The interface `ServiceConnection` is a callback interface. It is used to provide the instance which called the service with the information, when the

connection to the service was established or canceled. When connected to the service, the binder object for the communication is committed here.
Data can be requested from or given to the service with the help of a binder. If a developer wants to enable a data communication with the service, he has to develop own binder implementations and return them to the method `onBinder(Intent)`. This is started every time `bindService()` is called.
By convention, the communication protocol should be written in the Android Interface Definition Language (AIDL), but this is not mandatory.
To end the connection to a service, the method `unbindService(Intent)` is called with the connection to end as a transfer parameter. If no connections to the service are left, the method `onUnbind()` is called to have the opportunity to react appropriately.

3.8 Intents

In this section intents are explained in more detail. As described before, intents are necessary every time one component wants to start another. The basic idea is that all components should be reusable. Instead of calling an activity explicitly, an intent is called and told which kind of task it is supposed to do. The system listens to the intents, searches for a matching component in all applications and starts it. In this way a repeating implementation of often used standard functionalities can be avoided. Besides, an explicit calling of every activity would be unreasonable and impossible for the developer. It might be conceivable in case of system functionalities, but in case of post installed additional software it is virtually impossible.
Intents are always handed to a context. This interface provides all information of the application surface. All components are subclasses of a context implementation. Elements of this user interface need to know in which context they are indicated, too. Via the context, any communication with the system, the internal services and global settings is performed.
Basically, there are two kinds of intents: so called explicit intents, to which the name of the component to start is given and implicit intents, to which only the description of the task is given. Based on that, the system chooses an appropriate component.

To use a particular component, the developer has to use an explicit intent. The Intent is told which component exactly it has to call. A component is chosen by its class name, which is either given to the constructor `Intent(Context, Class)` or to the method `setComponent(ComponentName)`. In this case, `ComponentName()` consists of a package and a class. The context inside the constructor also serves to determine the package of the component.

If the component needs additional information, the intent can be extended by a data bundle. This bundle can manage simple key-value-pairs, where the values can be either primitive data types or serializable objects. This limitation on serializable objects is necessary, because bundles can also be applied to exchange data.

Explicit intents are used mainly to start a new component within an application, for example to open a new window where the user can enter data which is passed to the called activity. But if a developer wants to manage specific data, e.g. contact data, web pages or phone calls, an explicit intent does not work anymore, because he does not know the specific class.

When it comes to implicit intents, the system decides, which installed component matches the developers demand best. Therefore, necessary data is described in two parts: At first, the intent gets a `String` object, where the executing task is described, second, via a Uniform Resource Identifier (URI) the data afflicted by the task is given. Additionally, the task can be specified further by giving a category or giving additional information via a bundle.

Some standard activities are predefined by the API. These include widely used activities, e.g. displaying, changing and sending data. If a developer defines his own activities for his component, he has to make sure that their names are unmistakable.

The structure of the URI depends on the type of data. Usual URIs are phone numbers (tel:), web pages (http: and ftp:), e-mail addresses (mailto:) or contact data of the device (content://contacts/people).

An intent can be described more precisely by the category. Hereby, amongst others, a declaration can take place, if a called component is only used for testing or if it is supposed to run while the device is docked at a docking sta-

tion. Like the actions, predefined categories are provided by the operating system. It is also possible to give two or more categories to an intent.
The operating system is responsible for solving an implicit intent and determining the matching components. The package manager analyses action, type and category and compares them with the data noted in the manifest.

3.8.1 Intent filter

It has to be declared in the manifest, which intents a component has to react on. Basically, a component can always be called by an explicit intent, but for implicit intents a filter has to be defined. This happens inside the activity element in the manifest.
Any number of intent filter tags can be defined inside the activity element. In it the elements action, category and data are stated, which correspond to the appropriate statement in the intent. Every intent filter is checked independently of each other.
Each filter has to react to at least one or more action. Only if an action tag in the filter and an action in an intent coincide, the component is qualified to run. After the action, the categories of the intent and the filter are compared. Both can contain any number of categories, but the categories of the intent have to be a subset of the categories defined in the filter. Does the intent contain a category not listed in the filter, the filter does not match.

3.9 Data storage/Content provider

There are different ways to safe data in Android. Just like in usual Java applications, it is possible to get access to the file system. In addition, the Android SDK includes a version of SQLite, a single user database management system. And third, the system of shared preferences exists especially for Android, which allows saving data via a framework. All three capabilities have advantages in different situations.
A good way for saving small but also complex data is the access to the file system. Data generated by an application is always stored in a directory and by default only visible for this application. To get access to these data, the methods `openFileOutput(String)` and `openFileInput(String)`, which expect the name of the data as object, are used.

With a flag, committed to the output-method, the data can be adjusted, if it is visible only for its own application or also for external programs.
Android provides methods to use an external storage system to save files. Because external storage devices are usually removable, Android first has to check the status of the external device before the application can get access to it. Additionally, Android does not allocate any visibility or safety maintenance for external storage systems. Any file on an SD-card is always readable by any application. However, the application has to examine every time explicitly if it has access to the device anyway.
Larger amounts of data should be saved in a data base. Android allocates a data base management system by default with SQLites, which supports mostly the SQL92 standard. Contrary to other database systems, this system is only meant for single user utilization. That is why no rights management via database management system takes place. However, that is not necessary, because every data base is exclusively applied to only one application. A direct data exchange between several applications is not provided.
Nevertheless, no data base is used in this project, therefore we omit a closer introduction. A description is found in [And05].
For saving smaller amount of data, Android provides so called shared preferences. These are managed by the system and can be processed by the developer via corresponding objects without taking care of the actual saving process. Like the name indicates, shared preferences first of all serve to save program settings. Although simple data can be saved here, shared preferences are limited to primitive data or strings. They are not qualified to save objects.
Application wide saved data can be requested via the context of the application with the method `getSharedPreferences(String, int)`. The retrieved object provides a value to a specific key.
To change data of the shared preference an editor is needed. This editor has a matching put method for any type of data and can also delete keys. Changes have to be established by a commit call.

3.9.1 Content provider

A special component for exchanging data in Android is the content provider. This allows exchanging data between applications in a standardized way. It

abstracts the underlying data storage and offers a consistent interface for the query of data. If an application wants to use a content provider, like other components, it has to be declared in the manifest.
The actual communication with the provider is not direct. Instead a content resolver with the required skills to communicate with different processes is used. The content providers' data is allocated in form of a precisely specified Uniform Resource Identifier (URI), where the pattern has to be of the type content. There are, similar to the SQL-syntax, create, read, update and delete (CRUD) methods to paste, retrieve, delete and change data.
Since content providers are not used within this project, a closer look at the syntax for those operations is waived. They are similar to the SQL-syntax with additional conditions. More precise information can be found in [And06].

3.10 Resources

To improve the management and maintenance of the application, Android offers the possibility to outsource static contents and save them in separate files. This data is stored inside the application in the *res* folder.
Basically, Android distinguishes between different types of data, which are stored in different subdirectories. It is possible to store pictures (stored in *res/drawable/*), other binary data (*res/raw/*), fixed Graphical-User-Interfaces-Layouts (GUI) (*res/layout/*), menu structure (*res/menu/*) and basic data (*res/value/*). These basic data includes strings, arrays, colors and others. Except for the binary data, all other data is saved in own XML-files.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <string name="hello">Welcome</string>
    <string name="app_name">AccelerometerApp</string>
    <string-array name ="languages">
      <item>English</item>
      <item>Deutsch</item>
    </string-array>

</resources>
```

Listing 3.2: Resource types

Some standard entries for resource files are shown in Listing 3.2. Normally they are classified in files according to their type.
Android provides the ability to overwrite resources for different device configurations. In this manner a simple and effective way of changing basic settings of the device, e.g. language or display settings, is given to the developer. These alternative resources are stored in their own directory. For example, if an application is supposed to be available in more than one language, German and English, all apparent strings are outsourced into the `strings.xml` file in the value directory. The alternative configuration, in this case German, is stored in the folder value-de.
When the system evaluates these resources, it searches for the matching configuration at first and gets access to the standard configuration later. An Anglophone device for example would use the folder values-en at first. If it does not find an entry there, it searches in the standard folder values for another entry.
This resource specialization is not only confined to languages (the country-code is according to the International Organization for Standardization (ISO standard), it also includes for example the display size or resolution. The access to hardware components like the touchscreen or a foldout keyboard is also possible. A more precisely description can be found in [And04].

```
package com.strowski.accelapp;


public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {

        public static final int button=0x7f050001;
```

Listing 3.3: Auto-generated resource file R.java

The compiler stores a unique ID for every resource in a generated file. That file is called *gen/R.java* and contains a class for every kind of resource and

for every element of these resources an individual ID. It can be seen in Listing 3.3.

To use the outsourced resources within the source code, again the context of the application is used. The method `getResources()` generates an object with which the resources belonging to the application can be called. This object has a get method for every type of resource, to which the ID of the requested resource can be passed.

It is also possible to get access to other resources already inside the resource. Such a reference always starts with an “@”, followed by the package name plus colon if the resource is outside the own application. After that, the type of resource and the name follows, for example `@string/app_name`.

3.11 Views

The illustration of a user interface in Android takes place via objects derived from the class `View`. These objects can be divided into two groups: At first the view components, which have an actual display, and second the view groups, which are responsible for the layout and contain other views. Overall, the concept is, despite an own hierarchy, quite similar to Java.

The Android API has, similar to the Java Development Kit (JDK), several different types of layouts. These are the `FrameLayout`, which maintains several components of the display in a stack, the `LinearLayout`, which arranges components strictly horizontally and vertically, and the `RelativeLayout`, which arranges the components as a function of each other. The positioning of a child element inside a layout takes place by layout parameters. Every class defines its own inner class `LayoutParams`, which needs, depending on the layout, different parameters.

Simple components act like normal AWT- or Swing-components. They are defined by size, position and, if possible, by inner- and outer distance. Together with known views like buttons or text fields there are a lot of predefined views for charts or lists. All views are always created to a specific context.

Which view an activity is supposed to illustrate, is determined by the method `setContentView(View)`. Basically, the attribute can be a view or a view

group. Additionally, with the method `addContentView()`, an activity can be given more than one `ContentView`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:gravity="center"/>
<Button
     android:text="Refresh"
     android:id="@+id/button"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:textStyle="bold"
     android:textSize="25dp">
     </Button>

</LinearLayout>
```

Listing 3.4: Layout in XML

Views do not have to be built inside the source code. Default layouts for example can be predefined as XML resources. Both single components and larger hierarchies can be generated, and additionally both can be added in the source code.

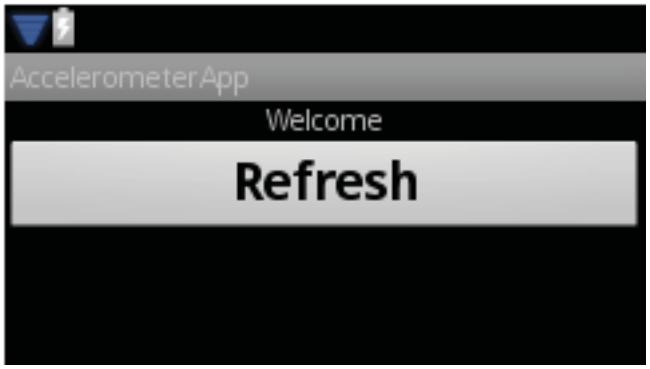

Image 3.4: Layout defined in listing 3.4

Listing 3.4 describes a simple layout with a text view and a button, the appearance on screen is shown in Image 3.4. XML elements are always named after the class they represent. Every element has a number of attributes which affect the layout. The attributes `android:layout_width` and `android:layout_height` for example define the width and height. Other attributes are named that clearly, too. The access to other resources via the @-notation is also shown in the listing. Especially the attribute `android:id` is important. With this attribute individual components of a larger view can be given a unique name. In this way they can be referenced in other XML files or the source code. The "+" symbol shows that a new ID for a specific element is applied.

3.12 Other Android components

Next to activities, services and content providers there is another component in Android: the broadcast receiver. Since it is not used within this project, but still quite interesting, only a short introduction is given.

With a broadcast receiver an application can react on broadcast announcements from the system or other applications. Most broadcasts originate from the system, e.g. a broadcast announcing that the screen turned off or battery is low, but applications can also announce a broadcast, for example when an application downloaded some data which is then available for other applications.

Broadcast receivers neither have a user interface nor implement a lot of code. Their only task is to receive and work on broadcasts in terms of intents.

4. Introduction to the Human Activity Recognition App

4.1 Introduction

With the task to implement an application for Android that allows the recognition of human physical activities, the goal is clear, but the rage of functions of the program is certainly described inaccurately. However, the way to reach this goal is given.

Approximately, signals due to the movement of the user are received by the sensors of the smartphone. These have to be converted into a form the application can analyze. Furthermore, it would be preferable to save and archive the acquired data at a central location for further investigations. Since the user should have a benefit by the program, a visualization of the data is necessary.

To get more naturalistic results, the application runs parallel to the normal programs of the phone in the background. By so doing, the user is in no way limited to his/her usual use of the phone, but this also guarantees the most naturalistic results possible.

In this chapter the acquisition, processing and dispatching of the data from the sensors is shown. Therefore, the individual components of the program are explained, without going into detail on the implementation. Moreover the communication between the individual components is a subject. As Android puts a lot of emphasis on reusability of its components, they are implemented as encapsulated as possible and the communication between them is very important. Additionally, the used frameworks and classes of the different components are explained.

4.2 Application overview

The Human Activity Recognition Application helps recognizing the physical activity of the user by analyzing the signals emitted by the accelerometers of the smartphone. In further processes, this data is stored, visualized and send to a web server for more examinations. After the application is started, it can run autonomically in the background without influencing the usual use of the phone, but if requested, it can show the received data on screen. To send the data to a server, the user has to initiate the process.

The application consists of two main activities with a user interface and two services performing most of the relevant work in the background. After starting the application, the welcome screen appears. Here the user gets essential information how to use the application and its features. From there on, the user enters the menu, where the different features of the application can be accessed. With the first button, the ”Start App”-button, the essential work of the application is started. When pushed, a service is started that obtains data from the accelerometers and saves them for further use.

With the “Show data”-button the user can visualize the saved data of the accelerometers and, i.e. his movement. When activated, a second activity is started to read the saved file and visualize the data on screen. This works only, if the activity for obtaining data has been started before.

The third button, the “Send data”-button, is responsible for sending the received data to the server. When pushed, a service is started that searches for a wireless internet connection and starts sending the saved data.

The application is stopped when pushing the “Stop”-Button. This shuts down all background services and exits the application. When started the next time, the service to obtain data from the accelerometers has to be started again.

To leave the menu without ending the application, the back button of the phone is used. This exits the users interface but keeps the application running in the background.

4.3 Menu

The entry activity of the application is the menu, implemented in the class `AccelMenu`. This is also the first interface the user sees. The main task of the menu is to switch between the different activities and services of the application. With the help of intents, which are caused when the corresponding button is pushed, other activities are called.

The graphical surface is created by a custom 2D graphic. For drawing 2D graphics, Android provides custom graphic libraries found in the package `android.graphics.drawable`.

Basically, there are two options to draw 2D graphics. The first option is preferred for simple graphics that do not change dynamically. Those graphics are drawn into a view object from the programs layout and the drawing of the graphics is handled by the system´s normal view hierarchy drawing process.

Our menu is an example for drawing graphics this way. The second option for drawing dynamic graphics is shown later, when the visualization of the data is described.

The users interface is build by the XML-file `menu.xml`. It shows the name of the application and the four buttons “Start”, “Show data”, “Send data”, and “Finish”. The XML file is defined as layout for this activity in the `onCreate()`-method by calling `setContentView(R.layout.menu)`.

As sub class of the super class `Activity`, `AccelMenu` has to implement the methods of the super class. However, not all of them are necessary, but one that must be implemented in any case is the method `onCreate()`. This method is always called when an activity starts. In this case, as mentioned before, the layout is defined here. Additionally, the buttons of the menu are implemented here. They were initialized before, but in this method, they are given the ID of the XML file and connected with the `onClickListener`.

The `onClickListener` is an interface which invokes a callback method every time a view is clicked. Buttons are views, so every time the user clicks a button of the menu, `onClickListener` recognizes that.

The interface has only one abstract method, `onClick(View v)`. This function implements the reaction to a button being clicked. The attribute v needs to match the buttons. In simple if-clauses, the `onClick()`-methods execute code depending on what button is activated. In our case, mostly intents are given to reach other activities or services.

As described before, intents serve to call other activities or services. In this case, we use only explicit intents, as the called activities and services are all in the same application and the class names are known. When defined, the intent is already given the class it is supposed to call as parameter. In this way, to start the specific activity or service, only the method `startActivity(intent)` has to be called. To stop an activity or service, the same intent can be used to call for example `stopActivity(intent)`.

4.4 Get sensor data

The most important part of the application is the class `AccelGet`. Here the data from the accelerometers are read and saved.

The class `AccelGet` is a service which is started after the according intent from the menu is given to it. Because the service has to handle only one request from other classes, it is implemented as `IntentService` class. This has the advantage that only one callback method and a constructor have to be implemented. The constructor, which has the same name as the service, serves for handling the thread the process runs in. In our case, the thread of the service is the same as the one of the rest of the application, so this constructor is trivial.

The callback method however is very important. Its name is `onHandleIntent()` and gets the calling intent as parameter. Thus, the method starts working when the intent arrives.

Inside the method, the class `sensorManager` is implemented. To get data from the sensors of the phone, Android provides the super class `SensorManager`. By calling the method `Context.getSystemService()` with the argument `SENSOR_SERVICE` an instance of the super class is build. To get a particular sensor, in our case the accelerometer, the method `getDefaultSensor()` with the argument `TYPE_ACCELEROMETER` from the type sensor has to be called. The value is saved in a variable.

With the now defined sensor a listener must be implemented to receive notifications when sensor values change. As parameters, the listener needs a `SensorEventListener` object to listen to, a registered sensor, and a specified rate to receive signals from the sensors.

Two of the three parameters are already available for the listener. The sensor was registered with the `SensorManager` before, and the rate of receiving signals must be one of four default values given by the Android API.

The third parameter is the `SensorEventListener`. This is a public interface provided by the Android API to receive notification from the `SensorManager` when the sensor values change. The interface `SensorEventListener` has two abstract methods that must be implemented when it is used. One is `onAccuracyChanged()`, which handles changes of accuracy of the

sensors. As the accuracy is defined one time and never changed, this method does not do anything here. The other is `onSensorChange()`, handling changes of the sensor values, and is described later.

The service `AccelGet` runs in the background, as described before. Once activated, it collects data from the accelerometers as long as it is stopped by the user by pushing the “Finish” button in the menu. This means that the sensors are enabled the whole time which causes a huge consumption of energy.

Additionally, in some cases, the obtained data caused a stack overflow and the application ran out of memory. This happened in earlier stages of the development of this application. As a result of that, the frequency of receiving data from the accelerometers slowed down until it stopped. The reasons for that are hard to find, because all information according the memory usage provided by the debugging tool are deleted when the program is closed.

4.5 Draw coordinates

In the class `DrawCoordinates` the visualization of the data received and saved by the class `AccelGet` is implemented. The challenge here is to create an activity that draws data which is updated frequently.

As mentioned before, the library for drawing graphics is implemented in `android.graphics.drawables`. The activity of this class needs to frequently re-draw itself, because the accelerometer data is also obtained frequently. This requires the second option to draw 2D graphics in Android, which works as follows.

The graphic is drawn onto a Canvas, which works like an interface to the actual surface upon which the graphic is drawn. The actual drawing is performed by an underlying bitmap, which is placed into the window of the screen. To create a new canvas, a bitmap has to be defined first, on which the canvas can be drawn. To draw on the canvas, the callback method `onDraw()` needs to be called.

As the application does not need a significant amount of processing, the view can be drawn with a canvas in `View.onDraw()`. This has the advantage to get a predefined canvas from the framework where the drawing calls can be placed. With this approach, the `View` class has to be extended and the `on-`

`Draw()` callback method has to be called. This is where all the calls to draw something are performed. The method will be called by the Android framework every time it is necessary that the view draws itself. To request the view to be drawn, it has to be invalidated by calling the method `invalidate()`. This indicates that the view is ready to be drawn and Android then calls `onDraw()`.

4.6 Send data

The goal of this part is to establish an internet connection and send data received by the accelerometers and saved in the `AccelGet` class to a specific server. This task is implemented in the class `AccelSendData`. This needs a stable connection to the internet via the wireless internet connection of the phone, a stationary web server to upload the data and the actual data exchange.

The focus in this part will not lie on establishing an internet connection. This is part of the Java SDK and a lot of literature and examples can be found here. Instead the focus will be on exchanging data with a server.

Android devices do not have a permanent IP-address by which they can be reached like a server. This will change entirely when IPv6 is established and every web-enabled device is provided with a unique IP-address. Until then, the communication between two Android devices is a bit more complicated. However, for this project, in the first version only a one way connection from the smartphone to the server to send data is necessary.

On Android platforms the `HttpComponents`-library by Jakarta Commons is preinstalled. This package supports network programming. Jakarta Commons is a project of the Apache Software Foundation. These classes can be found in the package `com.apache.http`, respectively in the sub packages. Interesting for our project is mostly the sub-project `HttpClient`. The full description can be found in [Apa01].

To monitor network connectivity, Android provides several classes. We have to consider that network connections can easily collapse on a mobile device, for example when leaving a building. Since sending data can take some time, we have to react on such an interruption. In this project we use two Android classes helping to establish a permanent network connection.

The first is the package `android.net.ConnectivityManager`. This class runs in the background and monitors the state of the network. If a network connection is disconnected or another one available, it tries to establish a new one, for example by switching from UMTS to WIFI-connection. Additionally, it informs the application via broadcast intent. In this way, the application can react on a disconnected network connection by breaking the process and informing the user.

The `ConnectivityManager` returns objects from the type `NetworkInfo`. The package `android.net.NetworkInfo` holds several methods to get important information about the stage, the type and the name of connection. The class `AccelSendData` provides a service, that starts when it is called by an intent, which is send by the `AccelMenu` class when the corresponding button is pushed. It stops, when the complete file is send. The detailed implementation is described later in chapter five.

To establish a connection to the internet, the receiver of the smartphone has to be turned on the whole time. This consumes a lot of energy, which affects the battery running time and in this way the operating time of the smartphone.

Unfortunately, this class with all of its functions could not be tested in an appropriate way. At first, the project that implements a program on a web server to receive the data from our application is in a very early stage and not ready to receive the data yet. Second, the emulator of the Android SDK provides a network simulation to simulate those tasks. However, on the development computer this simulated network frequently collapsed once a connection between the simulated smartphone and the simulated server was established.

4.7 Finish

With the “Finish” button the application is stopped. All processes running in the background, receiving data from the sensors and saving them, are stopped, and the application is left. Therefore, it sends an intent by calling `stopService(Intent)` to the class `AccelGet` and the receiving of data from the accelerometers is stopped. When the application is started again, the process of collecting data has to be re-started with the start button.

4.8 Additional features

Some features of the application did not make it into the program introduced here. That is because the implementation was faulty or just did not worked in the way it was planed. However, these parts are mentioned here to get a connection for a further work on this project.

The first and obvious feature that does not appear in our application is the actual recognition of physical activity performed by the user. In earlier stages, the class `AccelMath` was supposed to retrieve the data from the accelerometers and apply several algorithms on it to calculate the performed physical activity. Those algorithms produced a stack overflow, which quickly leads to an abortion of the program. After this problem could not be solved in a adequate way, we decided to outsource the physical activity recognition to the web server, where more computing power is available.

Another feature tries to automate the data transfer to the server. The idea is that every time the smartphone is connected with the recharger, the service `sendData` gets active and sends the received data to the web server. This would reduce the energy consumption of the application, because the wireless internet connection does not have to be active the whole time, which consumes a lot of energy. While recharging, the application does not receive new data from the accelerometer, because the smartphone is not used, so no information of the user's activity is lost.

To receive information from the system like the connection to a recharger, Android provides the system of broadcast receiver as described above. When the service `sendData` receives such a broadcast from the system, it can automatically establish the connection to the internet and start uploading the file to the server. However, as the service to upload data could not be tested, this function could neither.

5. Implementation

5.1 Introduction

This chapter deals with the concrete implementation of the Human Activity Recognition Application. In the following paragraphs, the way of getting data from the accelerometers, adapting the data for visualization, saving, and sending via internet for further treatment is described gradually.

At first, the structure of the program, the available components and the special requirements are described. Furthermore, the static information of the manifest is characterized.

Subsequently, we take a look at the acquisition of data from the accelerometer and their saving, followed by a description of the visualization. At last, the connection to the server and the sending of the obtained data is described in detail.

5.2 Manifest

To have a general overview of the application, the manifest is treated first. In later paragraphs we can refer to it, when describing the important links between single activities. Consisting of only a few different layers, which are only used for internal matters, the description is kept short.

```
<?xml version="1.0" encoding="utf-8"?>

<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.strowski.accelapp" android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission andro-
id:name="android.permission.WRITE_EXTERNAL_STORAGE">
    </uses-permission>
     <uses-permission andro-
id:name="android.permission.INTERNET">
    </uses-permission>
    <application android:icon="@drawable/icon" andro-
id:label="@string/app_name">
        <activity android:name=".AccelMenu" andro-
id:label="@string/app_name">
```

```
            <intent-filter>
                <action andro-
id:name="android.intent.action.MAIN" />
                <category andro-
id:name="android.intent.category.LAUNCHER" />
            </intent-filter>
     <activity android:name=".AccelDrawCoordinates"
     android:label="@string/app_name">
        </activity>
        <service android:name=".AccelGet"
     android:exported="false">
        </service>
        <service android:name=".AccelSendData"
     android:exported="false"/>
     </application>

</manifest>
```

Listing 5.1 Application manifest

As seen in Listing 5.1, the application has two activities and two services, whereby the activity `AccelMenu` is provided with a matching filter to serve as entry point for the application. The attribute `android:exported` is used to declare that the service is only usable for this application.

Also, the permissions used are seen clearly. The permission `WRITE_EXTERNAL_STORAGE` allows access to the external SD-card of the phone in order to write data on it. To get access to the wireless internet connection for sending data, the second permission `Internet` is responsible.

5.2.1 Resources

Lots of the used data in the application are saved as static resources. In this manner simple enhancements and alterations can be made by changing the matching XML entry.

For example graphical layouts are saved as XML-document. They involve complete view components as well as the menu.

Furthermore, every text presented to the user is outsourced in an XML-document. In doing so, the application can be constructed multilingually.

5.3 Accelerometer

The main task of the application is to collect data from the phone´s accelerometers. As described above, this happens in the class `AccelGet`.

Also as described in chapter four, the method `onSensorChanged()` is responsible for that. The code is shown in Listing 5.2. As callback method from the interface `SensorEventListener`, it gets the data from a registered listener. Every time a change in the sensor values is registered, the method saves this value in a float variable according to its axis, by calling the field `event.values[]`.

```
public void onSensorChanged(SensorEvent event)

     {

          String output1 = Float.toString(x);
          String output2 = Float.toString(y);
          String output3 = Float.toString(z);
          String saveString =
               'x'+output1 + 'y'+output2 + 'z'+output3;

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

          saveData(saveString);

     }
```

Listing 5.2: Method `onSensorChanged()`

The parameter `event` is from the type `SensorEvent` and given to the method as argument. The class holds information of the sensor such as the type of sensor or the sensor´s data. Now the values of the accelerometer are available for further use.

Additionally, the float values are converted to a `String` to be saved in the method `saveData()`. In Android, only strings or bytes can be saved in a file this way. The file on which the data is saved and the path where it is stored have to be declared at the beginning of the class. It is important to mention that the application can save files only on the external storage device of the phone, so it is necessary to insert a SD-card.

The path `"/sdcard/_ExternalSD/file/data.txt"` is added to the file when it is created. This is shown in Listing 5.3.

```
File saveFile =

          new File("/sdcard/_ExternalSD/file/data.txt");
```

Listing 5.3: Declaration of the save file

The actual saving process takes place in the try block of the method. First, a new `FileOutputStream` is declared and given the saved file as a parameter. Additionally, a Boolean variable is given, to define if new data will be appended to an existing file or not. In our case, the parameter is set true, so the data will be appended. The created stream is given as parameter to the `OutputStreamWriter`, which writes the data to the specified file. The exceptions are necessary, in case the file or the directory to write in are incorrect or missing, for example when no SD-card is insert. The method `saveData()` is shown in Listing 5.4.

```
public void saveData (String Data)

     {
          try
          {
           FileOutputStream fOut = new FileOutput-
Stream(saveFile,true);
           OutputStreamWriter osw = new OutputStreamWri-
ter(fOut);

           osw.write(Data);
           osw.flush();
        }
          catch (IOException ioe2)
        {
           Log.i("data error", "" + ioe2);
               Toast.makeText(this, "IOExcep-
tion"+ioe2.getMessage(), Toast.LENGTH_SHORT).show();
        }
          catch (Exception e)
        {
           Log.i("file error", "" + e);
           e.printStackTrace();
               Toast.makeText(this, "Exception",
Toast.LENGTH_SHORT).show();

        }


     }
```

Listing 5.4: Method `saveData()`

5.4 Visualization

The visualization of the accelerometer data takes place in the class `AccelCoordinates`. This activity displays a coordinate system and shows the values of the three axes of the accelerometer in a graph.

As the class `AccelCoordinates` extends the super class `Activity`, first of all the method `onCreate()` has to be called. In distinction from the activity `AccelMenu`, the layout is created dynamically, as described before. This can be seen in the method `setContentView()`(Listing 5.5), were the parameter is not a reference to an XML file, but a new `SampleView` is created.

```
setContentView(new SampleView(this));
```

Listing 5.5: Creating the View

This `SampleView` is defined in the equally called class which, also as described above, extends the super class `View`. In here, all of the drawing methods for creating the graphical interface are implemented. Listing 5.6 shows the code.

At first, in the method `SampleView`, a new bitmap object, a new canvas object and a new paint object are initiated. The paint object holds all the style and color information about how to draw the graphic.

Second, the file with the accelerometer data is loaded into this class. To save memory and accelerate the process, the file is read and stored in a `BufferReader`. From there on, the file can be used. Again, an exception catches the case that the method cannot find the file in the given directory.

```
public SampleView(Context context)
          {
                super(context);

                mBitmap = Bitmap.createBitmap(
                          1088, 1920, Bit-
map.Config.ARGB_8888);
                mCanvas = new Canvas(mBitmap);
                mBitmapPaint = new Paint(
                          Paint.DITHER_FLAG);
                try
                {
                     f = new File(
                     "/sdcard/_ExternalSD/file/data.txt");
                     fr = new FileReader(f);
```

```
                    br = new BufferedReader(fr);
                }
                catch(FileNotFoundException e)
                {
                    e.printStackTrace();
                    exc="FileNotFoundException";
                }

    }
```

Listing 5.6: Method `SampleView()`

The second method is the preset `onDraw()`–method, displayed in Listing 5.7. This method must be implemented when extending a class from the super class `View`. Here all of the drawing methods are called.

At first, the canvas to draw on is connected to the bitmap and the paint objects created in the method `SampleView()`. This happens by calling `canvas.drawBitmap(mBitmap, 0, 0, mPaint)`.

The size of the canvas is defined by the two values `HEIGHT` and `WIDTH`. The values are delivered by the methods `getHeight()` , respectively `getWidth()` from the canvas class. When the canvas is set up, the axes are printed by calling the method `printAxis(canvas)`, which we cover later.

After the canvas is defined and the axes of the coordinate system are drawn, three if-clauses ensure that the saved file is read first before the data is drawn. Otherwise the program would stop and the application crashes. To prevent that, a Boolean value `fileRead` is defined which is set to `true`, if the file has been read. So if the file has not been read yet, the first if statement is fulfilled and the method `readFile()` is called. This method, as the name indicates, reads the file and sets the value `fileRead` to true. This method is described a bit later.

```
protected void onDraw(Canvas canvas)
            {
                  canvas.drawBitmap(mBitmap, 0, 0, mPaint);

                  HEIGHT = canvas.getHeight();
                  WIDTH = canvas.getWidth();

                  printAxis(canvas);

                  if(fileRead==false)
                  {
                  readFile();
                  }

                  if(fileRead==true && readyToDraw==false)
                  {
                  sortData(canvas);
                  }

                  if(readyToDraw==true)
                  {
                        drawCordinates();
                        readyToDraw=false;
                  }

            }
```

Listing 5.7: Method `onDraw()`

In the next step, the method `sortData()` sorts the data before it is ready to get drawn by calling the method `drawCoordinates()`.This method gets the values from the sort method and draws them on screen. After one value-triple is drawn, the Boolean variable is set back to false to get a new value-triple and the whole process starts again. In this way, every time a triple of x, y, and z coordinates is drawn.

The axes of the coordinate system are drawn on the canvas by the method `printAxis` (), shown in Listing 5.8. Here, predefined methods of the classes paint and canvas are used to define color, stroke and labels of the axes. Most of the methods are self-explanatory by their name, for example `mPaint.setColor(color.WHITE)` sets the background color of the canvas to white. To draw lines, the method `canvas.drawLine()` is called with input variables, that represent the start- and end-coordinates of that line. The first two are the starting, the second the ending coordinates. As usual, the method ends with the call `invalidate`() to signal that Android can recall this method.

```
private void printAxis(Canvas canvas)
            {
                  mPaint.setColor(Color.WHITE);
                  mPaint.setStrokeWidth(2);
                  mPaint.setTextSize(10);
                  canvas.drawLine(10, 170, HEIGHT, 170, mPaint);
                  canvas.drawLine(30, 175, 30, 40, mPaint);
                  mPaint.setStrokeWidth((float) 0.5);
                  canvas.drawLine(30, 140, HEIGHT, 140, mPaint);
                  canvas.drawLine(30, 110, HEIGHT, 110, mPaint);
                  canvas.drawLine(30, 80, HEIGHT, 80, mPaint);
                  canvas.drawLine(30, 50, HEIGHT, 50, mPaint);

                  invalidate();

             }
```

Listing 5.8: Method `printAxis()`

The values of the accelerometers are saved ongoing in a string variable in the method `saveData()` of the class `AccelGet`. To read them, the method `readFile()` reads the file saved in the `BufferReader` and writes it into a new `String`, which is given to the drawing methods. The while statement ensures, that the String of the saved file is only read if it is not `null`.

```
private void readFile()
            {
                  try
                  {
                        String lineA = new String();

                        while((lineA = br.readLine()) != null)
                        {
                              fileRead=true;
                              line=lineA;
                        }
                  }
                  catch(IOException e)
                  {
                  e.printStackTrace();
                  }

            }
```

Listing 5.9: Method `readFile()`

To sort the data read in the method `readFile()`, every single value is given its axis when saved. By doing so, the saved `String` consists of progressive x, y, z – couples. Now, the sort algorithm has to search for the letters x, y, and z and read the value behind it, where every time, it finds a new letter,

a new value begins. These new values are saved in a new variable and are ready to be drawn. When a triple-value pair is found and saved, the method sets another Boolean variable (`readyToDraw`) to true, so the next if statement of the `onDraw()`−method can begin. The code of the method `sortData()` can be seen in Listing 5.10.

```
private void sortData(Canvas canvas)
            {
                  String xCord = "";
                  String yCord = "";
                  String zCord = "";

                  for(int i=counter;i<line.length()-10;i++){
                        if(line.charAt(i)=='x'){
                              int j=i;
                              while(line.charAt(j)!='y'){
                                    j++;
                              }
                              xCord=line.substring(i+1, j);
                              xCordFloatEnd = new Float(xCord);
                              counter=i+1;
                              break;
                              }
                  }
            }

            readyToDraw=true;

            invalidate();
```

Listing 5.10: Method `sortData()`

At last, the method `drawCoordinates()` finally draws the sorted coordinates. In Listing 5.11 this is shown for the x-coordinate. First, color and stroke width are defined by the methods `.setColor()` and `.setStrokeWidth()` and the line is drawn by `.drawLine()`. Inside the brackets the position of the coordinate on the screen is calculated. The values of `xxCords` and `xxCordf` represent the x-axis intercept, whereby `xCordFloatStart` and `xCordFloatEnd` represent the values of the accelerometer converted to `float` variables. They have to be adapted to the screen size calculation as well. After the coordinates are drawn, the ending point is transferred to the starting point where the next value is drawn. This works similar for the y and z value.

```
public void drawCordinates()
            {
                  mBitmapPaint.setColor(Color.BLUE);
                  mBitmapPaint.setStrokeWidth(1);

                  mCanvas.drawLine(xxCords,170-
                              (xCordFloatStart*20),
                                          xxCordf,170-
                              (xCordFloatEnd*20),mBitmapPaint);
                  xxCords=xxCordf;
                  xxCordf=xxCordf+1;
                  xCordFloatStart=xCordFloatEnd;

            }
```

Listing 5.11: Method `drawCoordinates()`

5.5 Data exchange

To send data to a server, the `AccelSendData` class is called by an intent, when the “Send data”-button of the menu is pushed. The service is started similarly as service in the class `AccelGet` described before.

Inside the method `onHandleIntent()`, first of all the saved file with the accelerometer data is read. After that, the data is sorted and saved as individual coordinates just like in the class `AccelCoordinates`. The calculated float values are saved in an array list and given to the method `sendData()`. Here, the IP-address of the targeted server is defined by a couple of `Strings` and given to the method `HttpPost` from the package `org.apache.http.HttpEntity` as described in chapter four. In two try-catch statements, the method tries to reach the server and, when the connection is done, send the list of strings to it. When the transfer is completed, a message is shown and the service is shut down.

In the first try statement, the parameter `postParameters`, which includes the array list of the accelerometer values, is given to the `httpPost` object, which is send to the targeted URL address in the second try statement.

The code of the method `onHandleIntent()` can be seen in Listing 5.12 on the next page.

```
private void sendData()
      {
            final String mServerIp = "IP-Adress";
            final int mHttpPortNum = 1234;
            final String mUrlString =
                  "http://" + mServerIp + "+" +
                  mHttpPortNum + "/file/file";
            final DefaultHttpClient client =
                  new DefaultHttpClient();
            final HttpPost httpPost = new HttpPost(mUrlString);


            try
            {
                  httpPost.setEntity(
                              new UrlEncodedFormEntity(
                                          postParameters));
            }
            catch(UnsupportedEncodingException e2){}

            try
            {
                  client.execute(httpPost);
            }
                  catch(ClientProtocolException e1){}
                  catch(IOException e1){}
      }
```

Listing 5.12: Method `sendData()`

6. Conclusion

This project shows one possible way to obtain data from the sensors of a smartphone for detecting and monitoring human physical activities with an application for the Android operating system. It is shown that the hardware accomplishes the minimum requirements to obtain data from the accelerometers of the Smartphone which are necessary for the essential calculations. Also the visualization of the data is possible with the abilities of the device.
A stable connection to a stationary server via a wireless network to dispatch the received data is also possible but could not be test under the given circumstances.
Nevertheless, the hardware shows several weaknesses which stand in the way of a frictionless utilization of the recognition of human physical activities. One major problem is the duration of the device, which is mostly influenced by the battery. As mentioned before, the application consumes a large quantity of energy, especially by the wireless network receiver and the permanent access to the sensors. However, both problems can be solved not only by increasing battery life or memory, but also with improved programming.

6.1 Developing in Android

One focus of the given task was the use of the Android operating system as platform for the program. This platform is at first very suitable for this kind of project, because it provides a large package of tools and resources to the developer. However, this is because of the abundance of information both helpful and difficult.
One point of criticism is the emulator provided by the Software Development Kit. At first, sensor emulation is missing, which renders the usage of the emulator in this project impossible. All of the testing cases had to be done on the device itself. Although such an sensor emulator is developed in a private project, it was not compatible with the software development kit on the development computer.
Second, the simulation of a network of two emulators shall be mentioned. Although the opportunity to establish such a network is given and well documented by the developers, the realization on the development computer was hard to implement and failed, with the result that the testing of sending data to a server was not possible.

However, in comparison to the disadvantages, the platform has a lot of advantages, including the provided tools, resources and documentation. At first, the available API has to be mentioned. Its large range of functions allows the realization of countless different conceptions. Especially the access to the hardware´s resources and the developer's independence to use them is tremendous and easy to utilize. Also a good concept is the idea of outsourcing static data and reloading it in the program. Thus, the reusability is increased and the program can react easily on different configurations of a device.
Altogether, the platform allows very complex projects without artificially restricting the developer. This and the fact that, as opposed to Apple, every developer can program on this platform without being restricted by the manufacturing company, makes the Android operating system ideal for experimental applications and projects.

6.2 Future

Although the development of smartphones is quite young, their popularity with customers is huge. And since decreasing prices are assumed for the future, this popularity will grow and even more people will use one.
Even though smartphones will not be competitive with conventional computers in the near future, their technical equipment will proceed. However, the fast development and dissemination of high performance wireless networks and the progressing introduction of the new Internet Protocol Version 6 (IPv6) will make an excessive increasing performance unnecessary. A lot of applications will run on stationary servers, and the mobile device only has to run the internet browser.
Android as platform seems to be well prepared for this. Today, already the operating system as well as the development kit are supervised and maintained very well and are updated frequently. And the variety of software developers on the one hand and the producers of the hardware on the other hand will continue to ask for further development and innovations.
The development of human physical activity recognition software will benefit greatly from that. With a permanent connection to a server the results of the activity recognition could be reported immediately and in case of accidents or unusual outcomes appropriate measures could be introduced.

For the current application also several enhancements could be thought of. For example, as mentioned before, an algorithm, which can detect when the smartphone is connected to the charging device and activate the WIFI transmitter and send the data automatically, would reduce the energy consumption.

And of course other sensors or features of the smartphone can be added to the app. For example a pulse detecting device could be connected via Bluetooth to measure pulse, heart frequency and blood pressure.

List of figures

Listings

Bibliography

[And01] Android Open Source Project
Android Source Code
http://source.android.com
Retrieved July 2011

[And02] Android Developers
What is Android?
http://developer.android.com/guide/basics/what-is-android.html
Retrieved July 2011

[And03] Android Developers
API Reference
http://developer.android.com/reference/classes.html
Retrieved July 2011

[And04] Android Developer
Localization
http://developer.android.com/guide/topics/resources/localization.html
Retrieved July 2011

[And05] Android Developer
Data storage
http://developer.android.com/guide/topics/data/data-storage.html
Retrieved July 2011

[And06] Android Developer
Content Providers
http://developer.android.com/guide/topics/providers/content-providers.html
Retrieved July 2011

[Apa01] Apache Software Foundation
Jakarta Commons HttpClient
http://hc.apache.org/httpclient-3.x/
Retrieved August 2011

[Eur01] European Union
Population projections 2008-2060
http://europa.eu/rapid/pressReleasesAction.do?reference=STAT/08/119
Retrieved August 2011

[Eur02] European Union
The impact of ageing on public expenditure: projections for the EU25 Member States on pensions, health care, long-term care, education and unemployment transfers (2004-2050)
http://ec.europa.eu/economy_finance/publications/publication6654_en.pdf
Retrieved August 2011

[Fra01] Fraunhofer Portugal
Mover application
http://mover.projects.fraunhofer.pt/
Retrieved July 2011

[Fue01] Fuentes Vasquez, Alvaro
Android system architecture diagram
http://en.wikipedia.org/wiki/File:Diagram_android.png
Retrieved August 2011

[Gar01] Gartner
Gartner Says Worldwide Mobile Phone Sales to End Users Grew 8 Per Cent in Fourth Quarter 2009; Market Remained Flat in 2009.
http://www.gartner.com/it/page.jsp?id=1306513.
Retrieved August 2011

[Gar02] Gartner
Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010
http://www.gartner.com/it/page.jsp?id=1543014
Retrieved August 2011

[Gar03] Gartner
Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012
http://www.gartner.com/it/page.jsp?id=1622614
Retrieved August 2011

[Lib01] AndroLib
Accumulated number of applications and games in Android Market
http://www.androlib.com/appstats.aspx
Retrieved August 2011

[Ope01] Open Handset Alliance
Industry Leaders Announce Open Platform for Mobile Devices
http://www.openhandsetalliance.com/press_110507.html
Retrieved August 2011

[Sen01] Sensor Logger
Sensor Logger By Chris Smith
http://chris.smith.name/android/
Retrieved July 2011

Declaration of originality

I hereby declare that this thesis and the work reported herein was composed by and originated entirely from me. Information derived from the published and unpublished work of others has been acknowledged in the text and references are given in the list of sources.

(Place and date) (Signature)