Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Android Workout

Semester thesis

Adrian Hess

adhess@ee.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Klaus-Tycho Förster, Jara Uitto
Prof. Dr. Roger Wattenhofer

September 18, 2013

Abstract

Today's smartphone become more and more powerful. They are packed with a lot of sensors. These sensors can be used to monitor many aspects in people life. One of the most important aspects is sports, especially working out.

In this thesis we want to use the phone's sensors to monitor the workouts of the user. Different to other approaches, this Android™ application monitors stationary workouts like push-ups or crunches. An algorithm was developed which, with the help of the accelerometer data, is able to count the number of repetitions during the workout.

Aside of the counting of repetitions an extension was implemented which helps the user to improve his technique for some selected workouts. The extension uses the orientation sensor to monitor the user's posture. With the knowledge of the posture at the turning points of the workouts, the application is able to give appropriate hints.

Contents

Chapter 1

Introduction

1.1 Motivation

Sports and working out was and still is one of the most important aspects in many people life. But for some just doing sports is not enough, they would like to monitor what they are doing. There already exist many equipment to monitor sports and working out, but this equipment often serves a small range of use or is rather expensive. Since this equipment is mainly a sensor with a piece of software behind it, why not use a smartphone to monitor the workout. Today's smartphone become more and more powerful. A lot of sensor are getting packed into a phone and these sensors' accuracy has improved over the last couple of years. Also, the number of people who own a smartphone is increasing, so it would be nice to use the device one already has for monitoring workouts.

So the hardware would be there, but what is still missing is the appropriate software. There are already applications which use the location sensors for tracking running or similar activities, but there are applications missing for stationary workouts. With all the internal sensors of smartphones and a new piece of software it should be possible to monitor also these kind of workouts and even give hints about one's technique.

1.2 Related Work

There are already several applications available which tackles the field of workouts. These applications can be divided into three categories.

Database applications
: Applications of this category contain a database of workouts. There is a short explanation for each workout how it is done as well as a list of affected muscles. Additionally to the database, workouts can be added to a training plan, which keeps track of what workouts with how many repetitions have

been executed. The number of repetitions is put in manually by the user. A good example of such an application is *Barbell Gym Tracker*.[1]

Personal trainer application
These applications set the rhythm in which the user should execute the workout with an acoustical signal. These rhythms are specified for only a few workouts per application. The user can set how many repetitions and sets he would like to perform. *Fitness Flow* is an example application of this group. [2]

GPS-based application
GPS-based applications track the user's position during workout. This is usually used for workouts like running and cycling. With the help of the GPS signal, the application is able to give information about the covered distance, average speed and many other things and also shows the covered route on a map. The applications keep track of all finished workouts in a diary. A popular application of this type is *Sportstracker*.[3]

1.3 Goal

The goal of this thesis is to create an application, which automatically keeps track of chosen workouts. Since the chosen workouts are stationary and often executed indoors, the GPS Sensor is not of any use for detection. Therefore an algorithm should be developed to recognize the repetitions of the workouts with the help of the smartphones internal sensors, like accelerometer or orientation sensor.

Additional to the counting of repetitions, the application should also detect if a workout is done in a wrong manner. In the case of incorrect execution, the application should give the user acoustical hints how to improve the execution of the workout.

The content of this thesis is structured as follows. Chapter 2 gives an overview of the functionality of the application. This includes which workouts can be chosen as well as the users customization possibilities. In Chapter 3, the algorithm for detecting repetitions is explained in detail. The process how the application detects incorrect execution of workouts and giving appropriate hints is discussed in Chapter 4. At the end is a short discussion on what could be done to improve the application further.

Chapter 2

Application Overview

This chapter gives a quick overview of the applications functionality. In a first part, the workouts are introduced for which the application is designated for. In a second part, all the elements which can be controlled by the user are explained.

2.1 Workouts

For the purpose of this application five workouts are chosen. Figure 2.1 shows the workout selection menu with the chosen workouts.

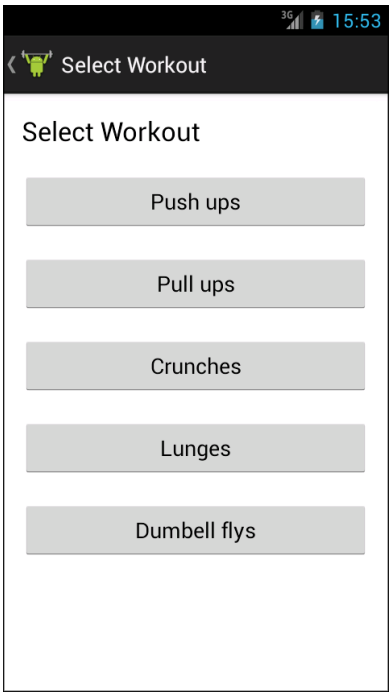


Figure 2.1: List of chosen workouts in the workout selection window.

These workouts have three things in common. First, they do not need any special equipment or any at all. In the case of dumbbell flys and pull-ups several other things can be used as a replacement for the dumbbells and grip, respectively. Second, these workouts do not require a lot of space, since they are stationary workouts. Third and the most important commonality, all the workouts have a start position, an intermediate position and an end position which is the same as the start position. This leads to an oscillating motion. The oscillating characteristic is important for the detection process, which will be explained in Chapter 3.

Additional to these commonalities, these workouts are very popular. At least the first three of the chosen workouts are known by almost everyone. This makes this application useful for a large user base.

2.2 User Interactions

In the following we want to explain what possibilities the user has, when he selected a workout.

2.2.1 Basic interactions

The purpose of the application is to monitor the workouts, but the application has yet to be able to detect when the workout starts. Therefore the application provides a *Start/Stop* button. When the *Start* button is pressed, the applications triggers a countdown of three seconds, which allows the user to get into starting position. The start of the monitoring is acoustically signaled. After the start, the same button transforms into the *Stop* button. With the *Reset* button the user can reset the counter to zero. This can be useful if the user wants to do multiple sets of the same workout. The application also provides detailed instructions of how the workouts are done. These basic interactions are the same for all workouts and are shown in Figure 2.2a.

2.2.2 Sensorposition

For every workout the user is able to choose from two predefined sensor positions, at which he should place the smartphone. The dialog box depicted in Figure 2.2b can be reached through the menu of the start screen from every workout. The first position is selected by default and does not have to be chosen before workout. As soon as the user clicks to change the sensor position, the application sets all the necessary parameters for detection and hints in the background.

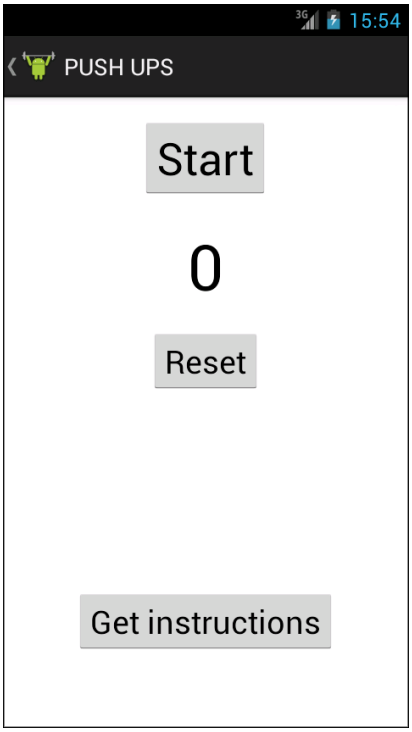
2.2.3 Detection parameters

Since the motion during the workout is not only dependent from the workout itself but also from the physical attributes of the user, the detection parameters may not fit properly for everyone. Therefore the user is able to adjust the parameters. The parameters which can be adjusted are the `TimeDiff`, which defines the minimum time between the starting/end position and the intermediate position during the workout and the `ValDiff`, which defines the minimum difference in acceleration between these positions. The use of these parameters are explained in Section 3.3.2. With appropriate adjustment of these parameters it should be possible for every user to detect the repetitions of workout.

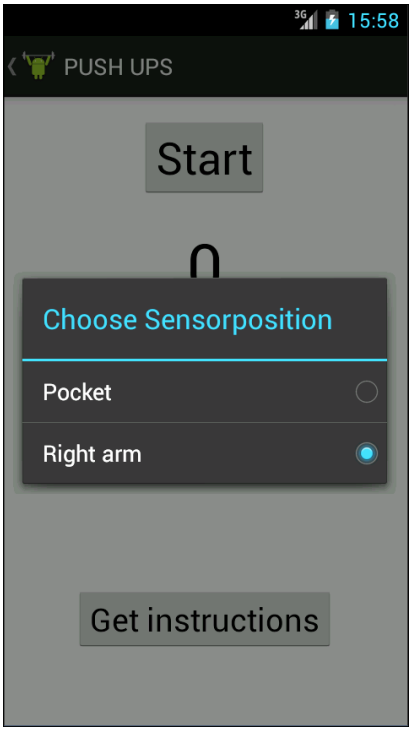
The slider for adjustment, as seen in Figure 2.2c, are shown as soon as the user selects *Custom parameters* in the menu. The limits for the slider are set to a reasonable range for every workout.

2.2.4 Give Hints

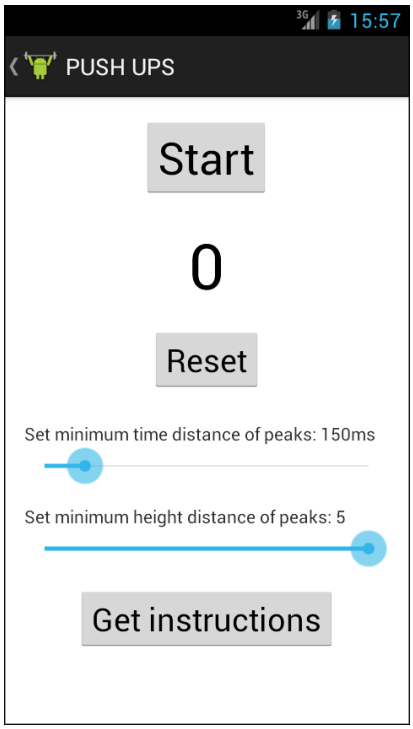
Finally, the user is able to choose if he wants hints to his execution of workouts. If the user enables this option, the *Hints for workouts* process described in Chapter 4 will be activated. As a result the user gets acoustical feedback, if he is doing the workout wrong.



(a) Basic interaction possibilities during workout.



(b) Dialog Box for sensor position selection.



(c) Slider for custom detection parameters.

Figure 2.2: Screenshots for user interactions.

Chapter 3

Detection of Workout repetitions

In this chapter, the process of how to detect repetitions of workouts is discussed. To perform such a detection, multiple tasks have to be performed. Figure 3.1 shows all the tasks that are required for the detection process. In the following sections, every task is explained in detail.

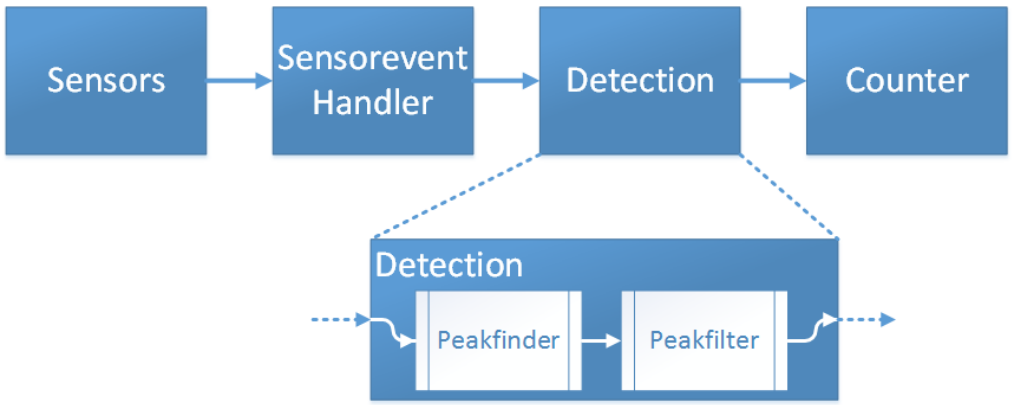


Figure 3.1: Flowchart of the repetition detection process.

3.1 Sensordata

Today's smartphones contain several motion sensors. These sensors can monitor three-dimensional device movement and positioning. Probably the best known and in this thesis for detection used motion sensor is the accelerometer. The accelerometer measures the acceleration applied to the device for all three axes. Figure 3.2 shows the three axes relative to the screen orientation. Since the axes changes with screen orientation, the application is locked to portrait mode only. This way the axes are relative to the physical device as shown in Figure 3.2.

The raw data supplied by the accelerometer are packed into a *sensor event*. As can be seen in Table 3.1, a *sensor event* contains the raw sensor data, origin of the data, timestamp and the accuracy of the data.

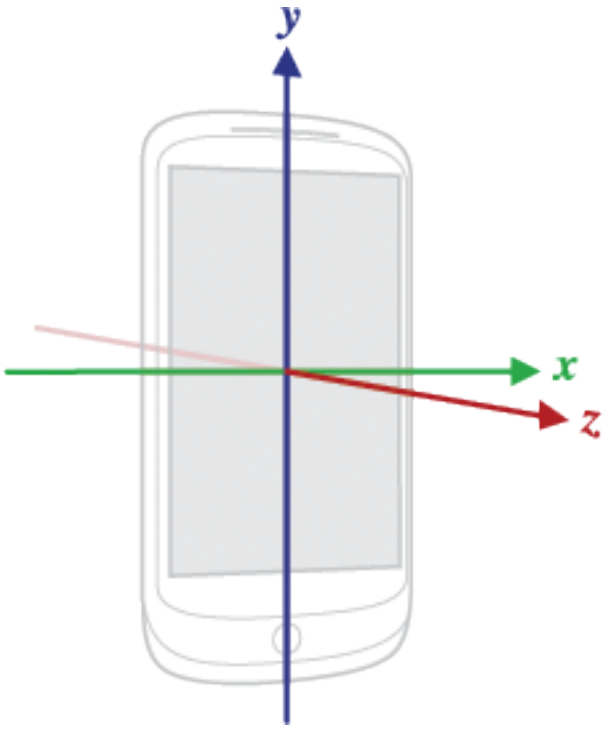


Figure 3.2: Coordinate system relative to screen orientation.[4]

Type	Variable name	Description
float[]	values	The length and contents of the values array depends on which sensor type is being monitored
Sensor	sensor	The sensor that generated this event.
long	timestamp	The time in nanosecond at which the event happened
int	accuracy	The accuracy of this event. (Low/Medium/High)

Table 3.1: Overview of a Sensorevent package.[5]

How often a *sensor event package* is generated by the sensor is an implementation choice. In our case, the sensor produces at least every 200 milliseconds such a package. Such a delay is small enough to detect the motion which belongs to the actual workout, but also high enough to get rid of most of the noise in the sensor signal.

3.2 Sensorevent Handler

The *sensorevent handler* is the first unit to process the data. It picks up the *sensorevent packages* as soon as the sensor has put one together. Since the *sensorevent package* contains more information than needed for the detection, the *sensorevent handler* filters out the necessary information. The needed information are the raw data in m/s^2 as well as the timestamp in milliseconds.

As mentioned in Section 2.1, the chosen workouts show an oscillating motion in at least one of the three axes. The *sensorevent handler* therefore only selects the value of the axis which shows the strongest oscillation. Which axis this is has been predefined with experiments for each workout and sensor position. The selected raw data is added to the list of all previous raw data. The same thing happens to the timestamps. This two lists are forwarded to the detection task whenever a new value is added to the list. Figure 3.3 shows an illustration of the described process.

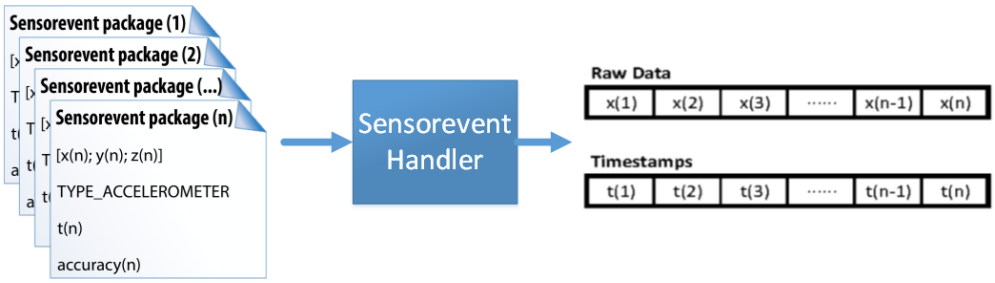


Figure 3.3: Input and output of the *sensorevent handler* task. In this illustration, the *sensorevent handler* chooses the x-axis for the detection task.

3.3 Detection

After the *sensorevent handler* passed all the necessary data, the actual detection can begin. The detection itself happens in two stages. The first stage is the *peakfinder*. After the *peakfinder* has done its job the *peakfilter* filters out the useful peaks used by the counter.

3.3.1 Peakfinder

The *peakfinder* algorithm goes through every value of the raw data list provided by the *sensorevent handler* and checks whether it is a local minima or maxima by comparing the value to its neighbors. If so, the value is stored into the new `peak value` list. At the same time, the corresponding timestamp is stored in the new `peak timestamp` list. All other raw data values and timestamps are

discarded. Therefore the algorithm produces two lists with peak values and the corresponding timestamps. The values are in time ascending order and always alternate from a minima to a maxima. Figure 3.4 shows an illustration what kind of lists get generated with the *peakfinder* algorithm. These lists are then forwarded to the *peakfiltering*.

Figure 3.5 shows a small example which values gets selected for a given list of raw data and timestamps.

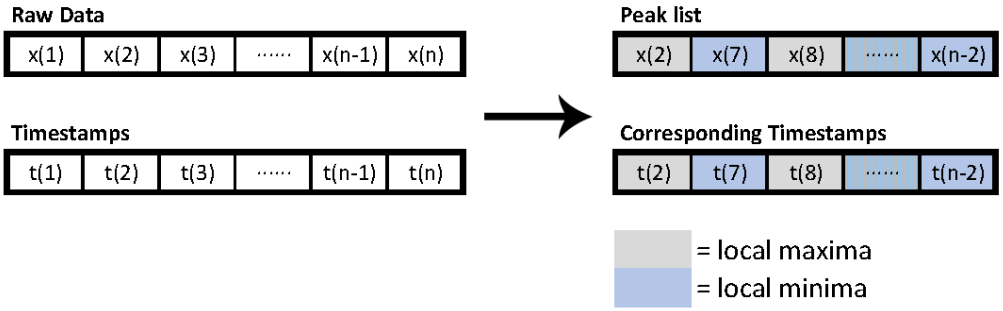


Figure 3.4: Illustration of the *peakfinder* task.

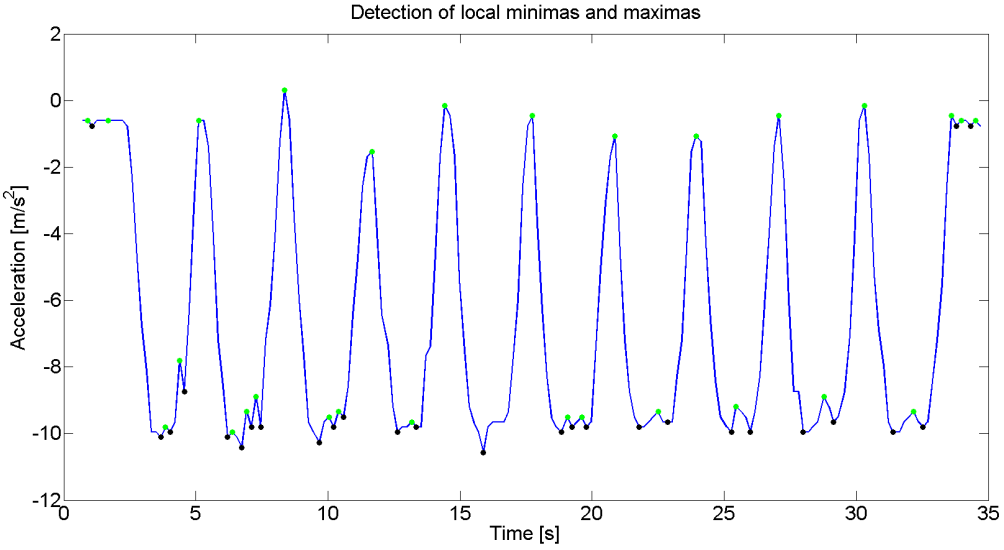


Figure 3.5: Example output of the *peakfinder* task in a graph, where the green dots are local maxima and the black dots are local minima. These values and the corresponding timestamps are sent to the next task.

3.3.2 Peakfiltering

After the *peakfinder* has found every local minima and maxima, the *peakfilter* algorithm filters out all the unwanted peaks in the `peak value` list. Unwanted peaks are caused either by the sensor's signal noise or by body shaking during the workout. Goal of the filtering is, that for every low and high curve of the oszilation one peak is detected. To achieve this goal, two sorts of filtering are used: *temporal filtering* and *acceleration distance filtering*.

The *temporal filtering* compares every two adjacent timestamps in the `peak timestamp` list. If the time difference is less than the specified parameter `TimeDiff`, then both timestamps are removed from the list. Together with the timestamps, the corresponding raw data from the `peak value` list are also removed. For each workout and sensor position the parameter `TimeDiff` was predefined after some experiments, but can also be customized as described in Section 2.2.3. With the removal of both of the adjacent values, it can be assured that local minima and maxima are still alternating. After this first filtering the timestamps are no longer needed.

The *acceleration distance filtering* compares every two adjacent raw values from the `peak value` list. Similar to the temporal filtering, it checks if the difference between the two values is smaller than the parameter `ValDiff`. If the difference is indeed smaller than `ValDiff`, than both values are removed from the list. After both filtering algorithms are finished, we have one list with the usable peaks only. This list is forwarded to the *counter*.

3.4 Counter

The *counter* is the final task in the detection process. It gets the filtered `peak value` list from the *detection* task and counts how many elements are in the list. Since there are still both local minima and maxima in the list, the number of elements have to be divided by two. Because we don't want to count the first half of a repetition, we have to round down the outcome of the division accordingly. The *counter* finally updates the number in the UI of the application and gives an acoustical signal to the user at the end of every repetition.

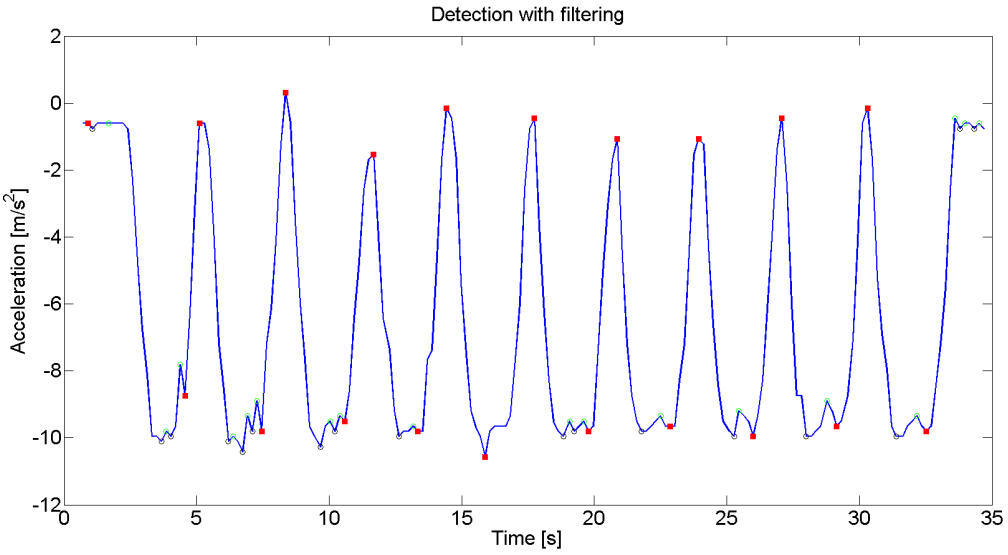


Figure 3.6: Example output of the *peakfiltering* task in a graph, where the red dots are the values which are sent to the counter. The green and black circles are filtered out.

Chapter 4

Hints for workouts

Alongside with the detection of repetitions for workouts, the application is able to give real time hints of how to perform the workouts. As for the detection process, the process for giving hints consists of multiple tasks. Some of these tasks run in parallel with the tasks of the detection process. Figure 4.1 shows the order of tasks which are required to give appropriate hints. Currently, hints are only implemented for the Push Ups and Crunches.

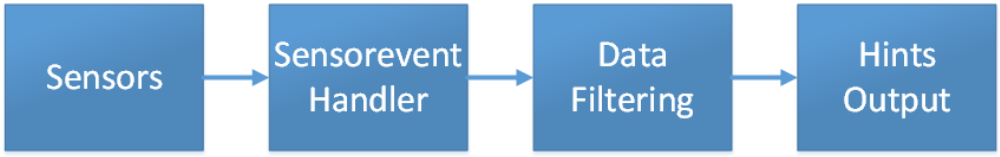


Figure 4.1: Flowchart of the Hints process.

4.1 Sensordata

In order to give hints, one does not only need to know the motion during the workout but also the position of the user relative to the starting position of the workout. Since the chosen workouts have an oscillating motion and the sensor are put into predefined positions, we have a rotation around one axis. We can measure this rotation with the help of the orientation sensor, which monitors the position of the device relative to the earth's frame of reference. The rotation axes are shown in Figure 4.2 and are defined as follows: [6]

- Azimuth: Degrees of rotation around the z axis. This is the angle between magnetic north and the device's y axis.
- Pitch: Degrees of rotation around the x axis
- Roll: Degrees of rotation around the y axis

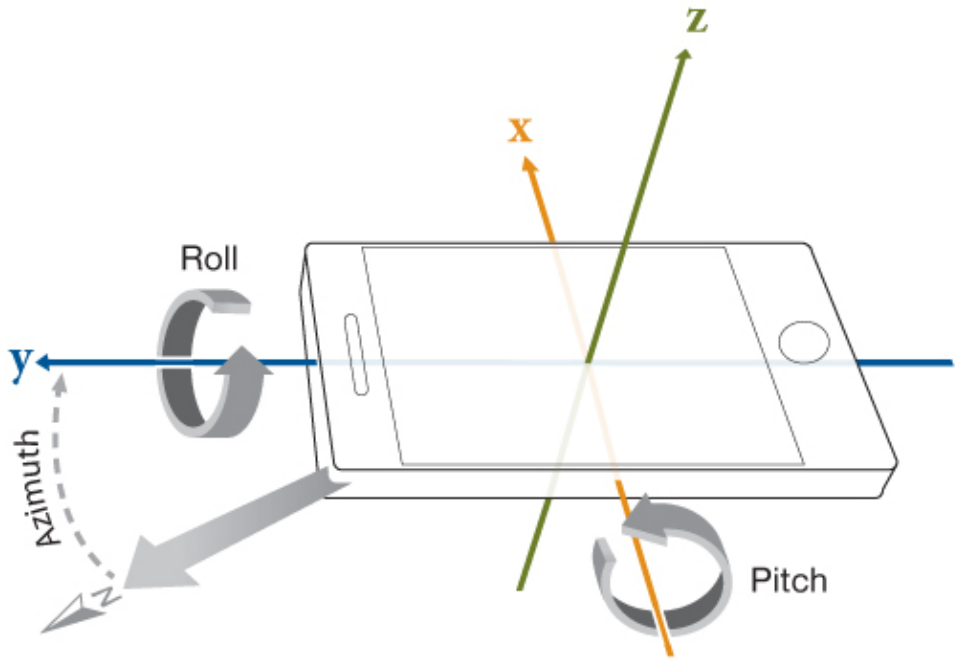


Figure 4.2: The three rotation axes relative to the device and magnetic north, respectively.[7]

As with the raw data from the accelerometer in the detection process, the orientation data are put into a *sensorevent package* and are produces at least every 200 milliseconds.

4.2 Sensorevent Handler

As described in Section 3.2, the *sensorevent handler* has to process all the *sensorevent packages* from the accelerometer. Additional to these packages it also have to process all the packages from the orientation sensor if the hints are activated. As for the acceleration data, not every information packed into the *sensorevent* by the orientation sensor is necessary for further tasks, therefore the *sensorevent handler* picks the value of the axis around which the device rotates during the workout and append this value to an `angle` list with all the previous values from the same axis.

Since the accelerometer and the orientation sensor don't necessarily provide *sensorevent packages* as often and at the same time, the *sensorevent handler* is also responsible for keeping the lists of the accelerometer and orientation sensor values at equal size. If one of the lists fall short by more than one value, the

sensorevent packages for the larger list gets dropped until the smaller list picks up with the larger one. It is crucial for the *data filtering* task that the two lists are of equal size.

4.3 Data filtering

The filtering process of the orientation data runs parallel to the filtering process of the acceleration data. Whenever a data point from the acceleration list is erased by the *peakfinder* or *peakfiltering* task, the data point at the same spot of the `angle` list also gets erased. As a result we get an angle of the chosen rotation axis for every peak in the detection process. Due to this filtering method it was necessary that the *sensorevent handler* kept the acceleration data and the orientation data of equal size. Because this way, it can be assured, that the acceleration data and the corresponding filtered angles are produced at almost the same time. Figure 4.3 illustrates how the angles are filter parallel to the accleration data in the detection process.

4.4 Hints output

The *hints output* task consist mainly of comparing two consecutive angles from the filtered `angle` list. Whenever the application detects a repetition it compares the last angle with its predecessor. Since the peaks alternate from a minimum to a maximum, the difference of the angles shows the rotation from starting/end position of the repetition to the intermediate position. With the help of this difference the application is able to determine if the user is doing the workout correctly or not. If the user is doing it right, the application gives the usual acoustical feedback, else the application will tell what has been done wrong. Two examples of what kind of hints the application can give with this method are shown in the next section.

4.5 Implemented hints

Currently there are two hints implemented for two different workouts. One is for the push-ups and the other one is for the crunches. These two examples explains the use of the orientation sensor for the purpose of giving hints.

For the push-ups the application is able to detect if the user is going deep enough, whereas for the crunches it is able to detect if the user is bending too less or too much. Figure 4.4 shows for both workouts at which point the angles get into the filtered `angle` list which is used for the difference measuring.

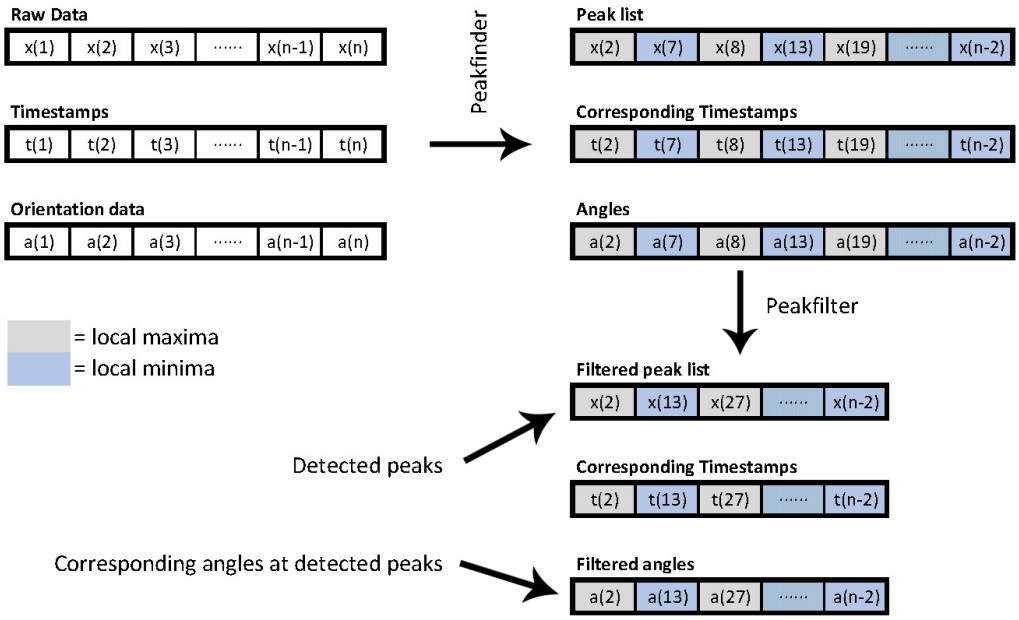


Figure 4.3: Illustration of the *data filtering* task.

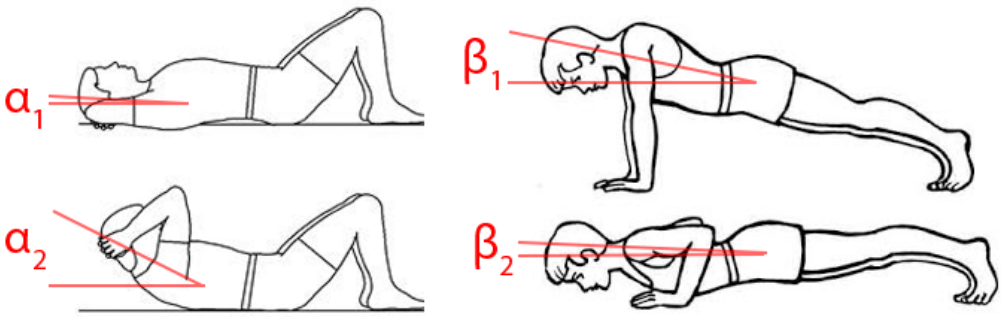


Figure 4.4: The used angles for hint detection for both crunches and push-ups.[8][9]

For both workouts the range of angle difference, at which the workout is executed correctly, has been predefined through experiments. If the measured difference is outside this range, the corresponding hint is given.

Chapter 5

Future Work

While writing this thesis, several ideas came to mind how to improve or extend this application. Unfortunately, the time for writing this thesis was limited. Therefore let us have a short look what could be done in the future.

5.1 Possible improvements

Sensor position independent
Currently there are two sensor positions for each workout which have to be set manually by the user. It would be a nice feature, if the application automatically detects the position of the phone. It could go even further that the application allow any random position of the phone and automatically sets the corresponding axes and parameters for the position.

Hints recognition improvements
The recognition process for the hints in the current state of the application is a very simplistic but also not very accurate approach. A more accurate approach would be needed to give better hints to the user.

5.2 Possible extensions

Hints
Until now there are only hints for two workouts. A possible extension is to implement more hints. These hints could either be for the other implemented workouts or additional hints for the workouts which already have hints.

Workouts
There are many known workouts, but the application is currently only able to detect five workouts. Therefore other workouts could be included in the application.

History

Since the application is able to detect the number of repetitions, it would make sense to implement a history function. Which stores all the repetitions in a database. The user can then see when he has done how many repetitions and how many repetitions he has done in total for each workout. The history would also show the percentage of repetitions, which has been executed correctly.

Workout plan

Similar to the database application described in Section 1.2, a workout plan could be implemented, where the user can decide when he wants to do which workout with how many repetitions.

Bibliography

[1] PalvajarviSoft: Barbell gym tracker. http://nosturitarkastus.fi/software/barbell/ Accessed September, 2013.

[2] Skimble Inc.: Fitness flow. http://www.skimble.com/ Accessed September, 2013.

[3] Sports Tracking Technologies Ltd.: Sportstracker. http://www.sports-tracker.com/ Accessed September, 2013.

[4] Google Inc.: Android sensor overview. http://developer.android.com/guide/topics/sensors/sensors_overview.html Accessed September, 2013.

[5] Google Inc.: Android sensorevent. http://developer.android.com/reference/android/hardware/SensorEvent.html Accessed September, 2013.

[6] Google Inc.: Android position sensors. http://developer.android.com/reference/android/hardware/SensorEvent.html Accessed September, 2013.

[7] Matlab Central: Rotation axes. http://www.mathworks.com/matlabcentral/fileexchange/40858-iphone-and-ipad-sensor-support-from-matlab/content/sensorgroup/Examples/html/CapturingAzimuthRollPitchExample.html Accessed September, 2013.

[8] Dispatch Online: Crunches. http://www.dispatch.co.za/losing-it/crunches/ Accessed September, 2013.

[9] NutriFit4Life: Push-ups. http://nutrifit4life.blogspot.ch/2012/09/spice-up-push-up-8-variations-to-try.html Accessed September, 2013.