Seminararbeit

Routentracker für Schweizmobil

Adrian Schmid - schmiad1@students.zhaw.ch

Zürich, 04.06.2014

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

Als begeisterter Velofahrer habe ich vor einiger Zeit die Tools (Website & Smartphone App) der Stiftung Schweizmobil für mich entdeckt. Diese Platform bietet den Benutzern offizielle Swisstopo Karten auf welchen alle offiziellen Velo-, Mountainbike-, Wander-, Inlineskate- und Kanurouten eingeblendet werden können. Zusätzlich kann man als registrierter Benutzer auf der Website eigene Routen zeichnen. So gezeichnete Routen können ausgedruckt, geteilt oder auf die Schweizmobil App übertragen werden. Was die Tools nicht bieten ist eine «Where have I been»-Tracker Funktion um gefahrene Routen direkt aufzuzeichnen.

Ich habe die Stiftung Schweizmobil bezüglich des geplanten Projekts und einer Schnittstelle zum automatischen Übermitteln von GPS Daten angefragt. Aktuell gibt es die Möglichkeit Tracks aus Files im GPX Format als Routen zu importieren. Schweizmobil habe bei der Implementierung seiner App einen «Where have I been»-Tracker angedacht jedoch aufgrund Zweifel bezüglich Genauigkeit und Batterielebensdauer wieder verworfen. Sie seien an den Ergebnissen meiner potentiellen Arbeit interessiert.

1.2 Ziele der Arbeit

Im dieser Arbeit geht es in einem ersten Schritt um das erstellen eines sehr technischen «Where have I been»-Trackers welcher unter anderem einen Export von gefahrenen Routen ins GPX Format beherrscht. Mit diesem Tracker werden dann in einem zweiten Schritt Experimente durchgeführt um eine möglichst hohe Genauigkeit bei möglichst geringem Akkuverbrauch zu erreichen.

1.3 Aufgabenstellung

- Einarbeitung in das Thema GPS, GPS Tracking, GPX Format und Dokumentation des Erarbeiteten
- Implementierung eines technischen «Where have I been»-Trackers zu versuchszwecken
- Implementierung des Datenexports ins GPX Format
- Durchführen verschiedener Tests bezüglich Akkuverbrauch und Genauigkeit
- Präsentation der Arbeit

1.4 Erwartete Resultate

- Dokumentation der Einführung (GPS, GPS Tracking, GPX Format)
- Implementation des «Where have I been»-Trackers inklusive Dokumentation
- Implementation des GPX Datenexports inklusive Dokumentation
- Dokumentation der Tests und Auswertung der Resultate
- Präsentation

1.5 Planung

Informationen beschaffen & Einarbeitung in GPS GPS Tracking, GPX Format	13.03.2014 – 20.03.2014
Erstellen eines Konzepts	20.03.2014 – 02.04.2014
Milestone 1: Konzept erstellt	**02.04.2014**
Analyse und Design der Software	03.04.2014 – 16.04.2014
Implementation	17.04.2014 – 07.05.2014
Testing / Dokumentation	08.05.2014 – 14.05.2014
Milestone 2: Implementation abgeschlossen	**14.05.2014**
Ergebnisse sammeln (Velofahren)	15.05.2014 – 01.06.2014
Ergebnisse vergleichen und analysieren	02.06.2014 – 05.06.2014
Dokumentation	06.06.2014 – 16.06.2014
Endkorrektur / Abschluss	16.06.2014 – 19.06.2014
Milestone 3: Arbeit fertig	**19.06.2014**
Abgabe der Arbeit	19.06.2014
Präsentation	19.06.2014

Tabelle 1: Projektplanung soll

Informationen beschaffen & Einarbeitung in GPS GPS Tracking, GPX Format	 13.03.2014 – 20.03.2014
Erstellen eines Konzepts	20.03.2014 – 02.04.2014
Milestone 1: Konzept erstellt	**02.04.2014**
Analyse und Design der Software	03.04.2014 – 16.04.2014
Implementation	17.04.2014 – 18.05.2014
Testing	18.05.2014 – 21.05.2014
Milestone 2: Implementation abgeschlossen	**21.05.2014**
Ergebnisse sammeln (Velofahren)	21.05.2014 – 01.06.2014
Ergebnisse vergleichen und analysieren	02.06.2014 – 05.06.2014
Dokumentation (inkl. Dokumentation der Implementation)	06.06.2014 – 17.06.2014
Endkorrektur / Abschluss	18.06.2014 – 19.06.2014
Milestone 3: Arbeit fertig	**19.06.2014**
Abgabe der Arbeit	19.06.2014
Präsentation	19.06.2014

Tabelle 2: Projektplanung ist

2 Grundlagen

2.1 Grundlagen der Erdvermessung

Um Karten oder Modelle der Erde zu erstellen, braucht es ein Vermessungssystem welches adäquat die Grösse und Form der Erde widerspiegelt. Wichtig beim Arbeiten mit einem solchen Modell ist, dass man sich bewusst ist, welche Annahmen getroffen wurden und welche Abweichungen aus sowohl der Messmethode als auch dem Modell entstehen. In diesem Kapitel geht es um die Klärung der Grundlagen die zum Verständnis der Funktionsweise des GPS notwendig sind. [14]

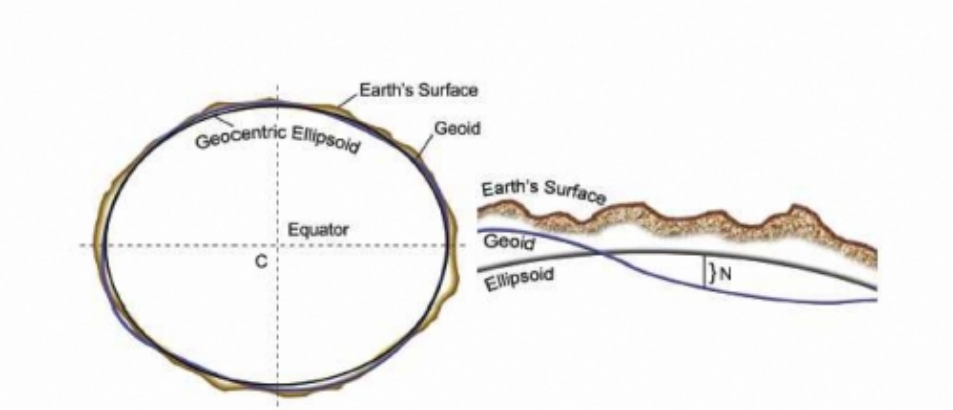

Abbildung 1: Verhältnis zwischen der Erdoberfläche, dem Geoiden und einem Referenzellipsoiden. [14]

Die Wissenschaft der Erdvermessung - auch Geodäsie - beschäftigt sich mit dem Studium von Grösse und Form der Erde, des Erdgravitationsfeldes und den Veränderungen des eben genannten. Zu den Grundlagen der Geodäsie gehört die Unterscheidung zwischen der physikalischen Erde, dem Geoid und dem Referenzellipsoid. Die physikalische Erde ist die Erde an sich. Der Geoid hat die Form, welche eine von Ozeanen bedeckte Erde, nur unter Einfluss von Gravitation und Rotation annehmen würde. Der Referenzellipsoid ist eine einfache, mathematische Form welche dem Geoid so gut als möglich entspricht. [14]

2.2 Georeferenzierung

Die Fähigkeit geografische Positionen genau zu beschreiben ist essentiell für sowohl Karten als auch geografische Informationssysteme. Diesen Prozess nennt man Georeferenzierung.

Eine geografische Position mithilfe von Längen- und Breitengraden auf dem Referenzellipsoiden zu beschreiben ist eine Möglichkeit der Georeferenzierung. Weitere Möglichkeiten wäre zum Beispiel die Verwendung von planaren oder kartesischen Koordinatensystemen. Im Umgang mit GPS werden ausschliesslich Längen- und Breitengrade verwendet, weshalb die weiteren Möglichkeiten im Rahmen dieser Arbeit nicht weiter erläutert werden.

Die Längen- und Breitengrade sind Winkelmessungen wischen dem Zentrum des entsprechenden Ellipsoiden und einem Punkt auf der Oberfläche. Breitengrade (in Englisch latitude) messen dabei den Winkel in Nord-Süd Richtung, Längengrade (in Englisch longitude) den Ost-West Winkel. [1]

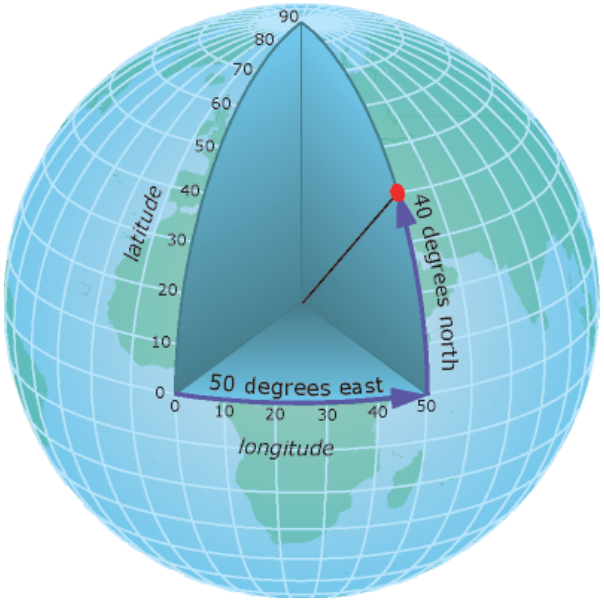

Abbildung 2: Lesen von Längen- und Breitengraden [1]

2.3 World Geodetic System 1984

Das World Geodetic System 1984 ist das geodätische Referenzsystem welches für GPS Positionsangaben verwendet wird. Als Koordinatenursprung dieses Systems dient das Massenzentrum der Erde.

Dieses System besteht aus:

- einem Referenzellipsoid für Ortsangaben nach geographischer Länge und Breite
- einem Geoid

- einem Satz dreidimensionaler Koordinaten der zwölf über die Erde verteilten Fundamentalstationen für die Verankerung der zuvor genannten Modelle in der Erdkruste [12]

2.4 Global Positioning System

GPS Besteht aus drei Segmenten, dem Raumsegment, dem Kontrollsegment und dem Nutzersegment. Das Raumsegment ist so konzipiert, dass es aus 24 bestehen soll, die die Erde in einer Höhe von rund 20200km alle 12 Stunden umrunden. Des weiteren ist das Raumsegment so angelegt, dass immer mindestens 4 Sateliten über einem Mindestelevationswinkel von 15° sichtbar sind. Jeder GPS-Satellit hat mehrere hochgenaue Atomuhren an Bord. Die Uhren arbeiten mit einer Grundfrequenz von 10.23 Mhz, die gebraucht wird, um das von den Satelliten gesendete Signal zu generieren. [2]

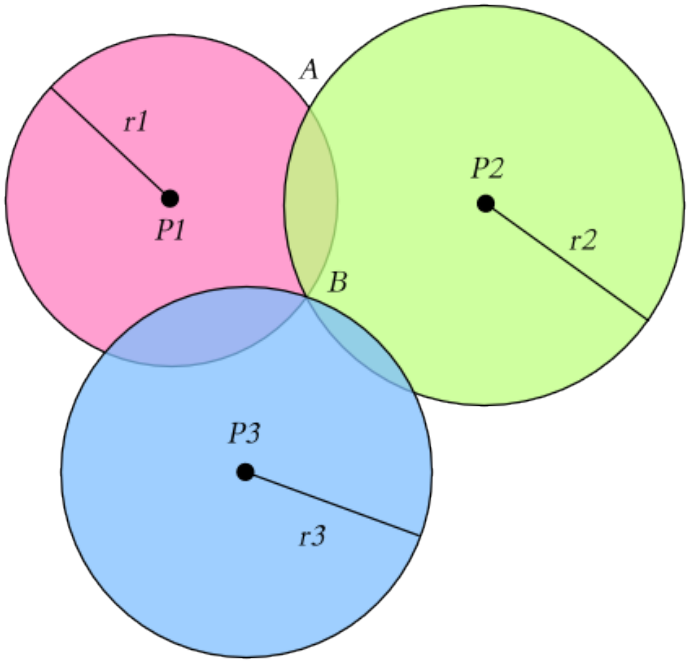

Abbildung 3: Trilateration [11]

Das Kontrollsegment besteht aus einer Hauptkontrollstation, der sogenannten Master Control Station (MCS), 5 Monitorstationen und 4 Telemetriestationen (Bodenantennen) verteilt über 5 Orte, die in etwa auf dem Äquator liegen.[2]

Das Kontrollsegment verfolgt die GPS-Satelliten, aktualisiert ihre Umlaufposition und kalibriert sowie synchronisiert ihre Uhren.[2]

Eine weitere wichtige Funktion ist, die Umlaufbahn eines jeden Satelliten zu bestimmen und seinen Weg für die nächsten 24 Stunden vorherzusagen. Diese Information wird jedem Satelliten eingespeist, um anschliessend von ihm gesendet zu werden. Damit ist der GPS-Empfänger in der Lage zu wissen, wo jeder einzelne Satellit erwartungsgemäss zu finden sein wird.[2]

Das Nutzersegment umfasst all diejenigen, die einen GPS-Empfänger einsetzen, um das GPS-Signal zu empfangen und so ihre Position und / oder Zeit zu bestimmen. Typische Anwendungen innerhalb des Nutzersegments sind die Navigation auf Land für Wanderer, die Bestimmung von Fahrzeugpositionen oder die Vermessung, die Navigation auf See, Navigation in der Luftfahrt, Maschinensteuerung usw. [2]

Die Positionsbestimmung per GPS basiert auf dem Prinzip der Trilateration. Der GPS Empfänger kennt die Position der GPS Sateliten und kann die Distanz zu jedem der Sateliten bestimmen. Wenn die Entfernung zu einem Sateliten bekannt ist, kann die eigene Position nur auf einer Kugelschale mit der Distanz als Radius und dem Sateliten als Mittelpunkt liegen. Durch den Schnitt dreier imaginärer Kugelschalen kann die Empfängerposition bestimmt werden. In der Abbildung 3 ist dieses Prinzip - der einfachheitshalber in 2D - ersichtlich. [2]

2.5 Distanzberechnung zwischen GPS Koordinaten

Zur Berechnung der Distanz zwischen zwei GPS Koordinaten wurde in dieser Arbeit die Haversine Formel verwendet: [16]

Definitionen:

$$\begin{array}{lcl} \varphi_1, \varphi_2 & = & \text{Breitengrade} \\ \lambda_1, \lambda_2 & = & \text{Längengrade} \\ \Delta\varphi & = & \text{Differenz zwischen den Breitengraden} \\ \Delta\lambda & = & \text{Differenz zwischen den Längengraden} \\ R & = & \text{Radius der Erde} \end{array} \tag{1}$$

Berechnung:

$$\begin{array}{lcl} a & = & \sin^2(\frac{\Delta\varphi}{2}) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\frac{\Delta\lambda}{2}) \\ c & = & 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)}) \\ d & = & R \cdot c \end{array} \tag{2}$$

Diese Formel verwendet anstelle des Ellipsoiden eine Sphäre was bei grossen Distanzen zu Fehlern führen kann. Bei den Distanzen zwischen Messpunkten von Velotouren oder

Wanderungen ist dieser Fehler nicht signifikant. Die Vorteile dieser Formel liegen in der einfachen Implementierbarkeit und der guten Performanz dieser Implementationen.

2.6 Das GPX Datenformat

Das GPX oder GPS Exchange Format ist ein XML Schema zur Darstellung von GPS Daten. Es kann verwendet werden um «Waypoints», «Tracks»und «Routes»darzustellen. Eine Ansammlung von «Waypoints»kann verwendet werden um eine Menge von Punkten darzustellen die in keinem sequenziellen Zusammenhang zueinander stehen. «Routes»und «Tracks»beinhalten sequenziell angeordnete «Waypoints». Der Unterschied zwischen den Beiden ist, dass bei einer «route»nur Wegpunkte bei Richtungsänderungen abgebildet sind, den Rest muss sich das System daraus berechnen. Ein «Track»beinhaltet alle aufgezeichneten Punkte. Im Rahmen dieses Projektes werden «Tracks»aufgezeichnet. [10] [17]

```
<?xml version="1.0" ?>
<gpx creator="ch.zhaw.gpstracker" version="1.0" xmlns="http://www.
    topografix.com/GPX/1/0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://www.topografix.com/GPX/1/0 http://
    www.topografix.com/GPX/1/0/gpx.xsd">
    <bounds maxlat="8.28439892" maxlon="8.28439892" minlat="8.28439892"
    minlon="8.28439892"/>
    <trk>
        <name><![CDATA[Baden-Oberrohrdorf]]></name>
        <trkseg>
            <trkpt lat="47.48865993" lon="8.22070903">
                <ele>379.0</ele>
                <time>2014-06-17T13:38:56Z</time>
                <sym>Dot</sym>
            </trkpt>
            <trkpt lat="47.48859126" lon="8.22059915">
                <ele>378.0</ele>
                <time>2014-06-17T13:39:04Z</time>
                <sym>Dot</sym>
            </trkpt>
            <trkpt lat="47.48381748" lon="8.28364061">
                <ele>410.0</ele>
                <time>2014-06-17T13:57:49Z</time>
                <sym>Dot</sym>
            </trkpt>
        </trkseg>
    </trk>
</gpx>
```

Listing 1: GPX Beispielfile

3 Umsetzung

Bei der Konzeptionierung und Entwicklung der Android App wurde Wert darauf gelegt, dass Programmlogik, Benutzeroberfläche, Exportfunktion und Datenbankanbindung soweit als möglich voneinander getrennt sind. Entsprechend sind diese Themen hier auch unabhängig voneinander beschrieben.

3.1 Programmlogik / Backend

Das Backend unserer Android Applikation besteht aus den beiden Klassen Track und Waypoint.

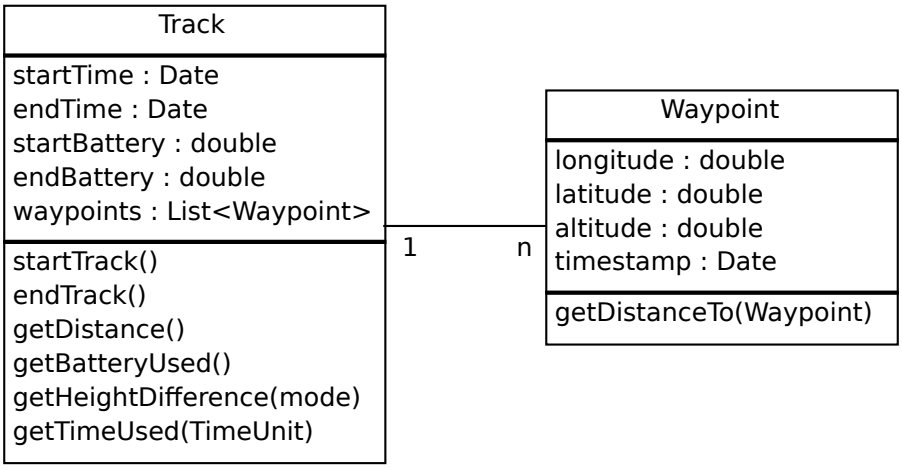

Abbildung 4: Klassendiagramm Backend

Im Klassendiagramm (Abbildung: 4) ist ersichtlich, dass ein Track primär aus einer Menge von Waypoints besteht. Zusätzlich Informationen zu Start- und Endzeit und Akkuzustand bei Start beziehungsweise Ende des Tracks mitgeführt. Die Waypoints bestehn aus den Koordinaten (Breiten- und Längengraden), der Höhe (Höhe über dem Referenzgeoiden) und einem Timestamp.

Der Track besitzt öffentliche Methoden zum Berechnen des Akkuverbrauchs, der Distanz, der Höhendifferenz und der Verbrauchten Zeit. Folgend wird kurz aufgezeigt wie die Funktionsweise dieser Methoden ist.

- getBatteryUsed() : $batteryUsed = startBattery - endBattery$
- getDistance() : Siehe Kapitel 3.3

- getHeightDifference(mode) : Siehe Kapitel 3.4
- getTimeUsed(TimeUnit) :
  - $timeUsedMs = endTime - startTime$
  - $timeUsedS = \frac{endTime-startTime}{1000}$
  - $timeUsedM = \frac{endTime-startTime}{1000*60}$
  - $timeUsedH = \frac{endTime-startTime}{1000*60*60}$
  - $timeUsedD = \frac{endTime-startTime}{1000*60*60*24}$

Weiter gibt es in der Klasse Track die beiden Methoden startTrack() und endTrack, welche das Tracking an sich steuern. In der Methode startTrack (Listing **??** werden zuerst die beiden Einstellungen mithilfe des Android PreferenceManager ausgelesen. Als nächstes wird die StartZeit und der Batteriezustand beim Starten des Tracks festgehalten. Zum Schluss wird auf den LocationManager mithilfe des per Parameter übergebenen Contexts zugegriffen. Diesem LocationManager wird dann ein neuer LocationListener angehängt.

Beim Anhängen des LocationListeners werden folgende Paramter mitgegeben:

- provider: Der Verwendete Provider für die Location Updates (in unserem Falle GPS)
- timeInterval: Das Zeitinterval. Wie oft soll der LocationListener angestossen werden?
- minDistance: Die minimale Distanz zwischen der aktuellen Location und der letzten. Wenn nach Ablauf des Zeitintervals diese Distanz nicht überschritten ist, wird der LocationListener nicht angestossen. [4]

,

```
1  private LocationListener = new LocationListener() {
2    public void onLocationChanged(Location location) {
3      // Called when a new location is found by the network location
       provider.
4      trackLocation(location);
5    }
6  }
7
8  public void startTrack(Context context) {
9    //Read Settings
10   SharedPreferences sharedPref = PreferenceManager.
       getDefaultSharedPreferences(context);
11   long minDistance = Long.parseLong(sharedPref.getString("
       min_dist_between_tp", "10"));
12   long timeInterval = Long.parseLong(sharedPref.getString("
       max_time_between_tp", "10000"));
```

```
13
14    //set meta infos
15    startTimestamp = new Date();
16    startBatteryPercentage = getBatteryStatus(context);
17
18    // Acquire a reference to the system Location Manager
19    LocationManager locationManager = (LocationManager) context.
        getSystemService(Context.LOCATION_SERVICE);
20
21    // Register the listener with the Location Manager to receive location
        updates
22    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
        timeInterval, minDistance, locationListener);
23  }
```

Listing 2: startTrack Methode

Das auslesen des Batteriezustandes geschieht dabei in der privaten Methode getBatteryStatus , welche im Kapitel 3.5 weiter erläutert ist.

Das Abschliessen eines Tracks ist in der Methode endTrack abgebildet. Als erstes wird der verwendete LocationListener vom LocationManager getrennt. Danach wird geprüft, ob Wegpunkte aufgezeichnet wurden, falls nicht wird false zurückgegeben. Wenn Wegpunkte vorhanden sind, werden Zeit und Batteriezustand bei Ende des Tracks (quasi jetzt) auf das Objekt geschrieben und das Objekt wird in die Datenbank geschrieben. Die Datenbankanbindung wird im Kapitel 3.2 weiter erläutert.

```
1  public boolean endTrack(Context context) {
2    LocationManager locationManager = (LocationManager) context.
       getSystemService(Context.LOCATION_SERVICE);
3    locationManager.removeUpdates(locationListener);
4
5    if (waypoints == null || waypoints.size() == 0) {
6      return false;
7    } else {
8      endTimestamp = new Date();
9      endBatteryPercentage = getBatteryStatus(context);
10
11     TracksDataSource tds = new TracksDataSource(context);
12     tds.open();
13     tds.createTrack(this);
14     tds.close();
15
16     return true;
17   }
18 }
```

Listing 3: endTrack Methode

3.2 Datenbankanbindung

Zum persistieren der Backend Objekte (Waypoint und Track) wurde das - bei Android Applikationen übliche - Content Provider Pattern [3] zur Ansteuerung einer SQLite Datenbank verwendet.

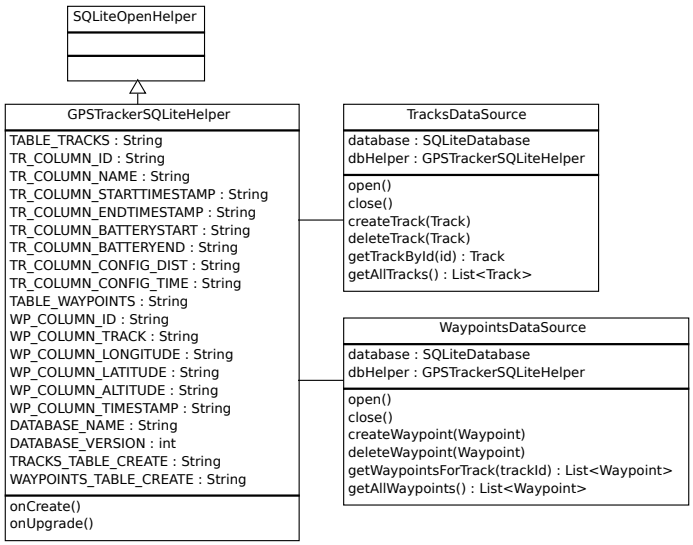

Abbildung 5: Klassendiagramm Datenbankanbindung

Zum Initialisieren der Datenbank und als Speicherort für die verwendeten Spalten- und Tabellennamen wurde der GPSTrackerSQLiteHelper verwendet. Dieser Helper ist von der Klasse SQLiteOpenHelper abgeleitet und überschreibt deren Methoden onCreate() und onUpdate(). Die erstere wird aufgerufen, wenn die Datenbank noch nicht auf dem Device vorhanden ist. Die letztere wird aufgerufen, wenn die DATABASE_VERISON höher ist als diejenige, die beim erstellen der Datenbank auf dem Device aktuell war.

TracksDataSource und WaypointsDataSource bieten neben den *open*() und *close*() Methoden, welche eine Verbindung zur Datenbank herstellen und die Verbindung wieder trennen, ein Subset der üblichen CRUD [8] Methoden, um die notwendigen Datenbankzugriffe auf eine möglichst einfache Art und Weise zu Managen. Im Listing 4 ist an einem Beispiel ersichtlich, wie die TracksDataSource Klasse verwendet wird.

```
1 TracksDataSource tds = new TracksDataSource(context);
2 tds.open(); //Connect database
3 tds.createTrack(track); //create track
4 tds.close(); //Close database connection
```

Listing 4: Beispiel: Track speichern

3.3 Berechnung der Distanz

Die Distanz zwischen zwei Wegpunkten wird gemäss der Haversine Formel (Siehe Kapitel 2.5) berechnet. Implementiert ist dies als Methode getDistanceTo(Waypoint) welche die Distanz zwischen dem aufrufenden und dem per Parameter übergebenen Waypoint wie in Listing 5 ersichtlich berechnet. Die Distanz eines ganzen Tracks entspricht der Summe der Distanzen zwischen allen Wegpunkten und ist in der Methode getDistance() der Track Klasse implementiert.

```
1  private final static int R = 6371;
2  public double getDistanceTo(Waypoint other){
3    double loThis = Math.toRadians(this.getLongitude());
4    double laThis = Math.toRadians(this.getLatitude());
5    double loOther = Math.toRadians(other.getLongitude());
6    double laOther = Math.toRadians(other.getLatitude());
7
8    double deltaLa = laThis-laOther;
9    double deltaLo = loThis-loOther;
10
11   double a = Math.pow(Math.sin(deltaLa/2), 2) + Math.cos(laThis) * Math.
       cos(laOther) * Math.pow(Math.sin(deltaLo/2), 2);
12   double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
13   return = R * c;
14 }
```

Listing 5: Distanzberechnung

3.4 Berechnung der Höhendifferenz

Die Berechnung der Höhendifferenz (Listing 6) hat drei Verschiedene Modi. Der Grund dafür lässt sich am besten an einem Beispiel erklären: Wenn man mit dem Velo von Göschenen (Rund 1100 m.ü.M.) auf den Gotthard Pass (rund 2100 m.ü.M) und danach weiter nach Airolo (rund 1175 m.ü.M) fährt, ist die Gesamthöhendifferenz 75 Meter. Abhängig von der körperlichen Verfassung wird der Velofahrer die 1000m, die er hoch gestrampelt ist, trotzdem in den Beinen spüren und entsprechend möchte er eine solche Angabe auch in einem GPS Tracker angezeigt bekommen. Basierend auf dieser Anforderung wurden die folgenden drei Modi entwickelt:

- positive: Die Summe aller Teilstrecken-Höhendifferenzen die positiv sind. (Im Gotthard Beispiel wären dies die ca. 1000m Bergauf)
- negative: Die Summe aller Teilstrecken-Höhendifferenzen die negativ sind. (Im Gotthard Beispiel wären dies die ca. 925m Bergab)

- total: Die Summe aller Teilstrecken-Höhendifferenzen. (Im Gotthard Beispiel ca. 75m)

```
public double getHeightDifference(String mode) {
  if (waypoints.size() == 0 || waypoints.size() == 1) { //Cannot calculate
    height difference without waypoints
    return 0;
  } else {
    double sum = 0;
    for (int i = 1; i <= waypoints.size() - 1; i++) {
      double div = waypoints.get(i).getAltitude() - waypoints.get(i - 1).
   getAltitude();
      switch (mode) {
        case "positive": //only count positive height difference
          sum += div > 0 ? div : 0; break;
        case "negative": //only count negative height difference
          sum += div < 0 ? (-1) * div : 0; break;
        case "total": //count everything
          sum += div; break;
      }
    }
    return sum;
  }
}
```

Listing 6: Berechnung der Höhendifferenz

3.5 Akkumessung

Das auslesen der Batteriezustandes ist in der Privaten Methode getBatteryStatus des Tracks mithilfe des Intent.ACTION_BATTERY_CHANGED implementiert. Die Mtehode liest den Zustand der Batterie aus und gibt diesen als double zwischen 0 und 1 zurück. Mithilfe des Intents können sowohl level (Integer zwischen 0 und einem gerätespezifischen Maximum) als auch scale (Integer der dem gerätespezifischen Maximum entspricht) ausgelesen werden. Eine geräteunabhängige Angabe zwischen 0 und 1 berechnet sich daraus wie folgt: $batteryStatus = \frac{level}{scale}$ [5]

```
private double getBatteryStatus(Context context) {
  IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
  Intent batteryStatus = context.registerReceiver(null, ifilter);
  int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
  int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
  double batteryPct = level / (double) scale;
  return batteryPct;
}
```

Listing 7: Auslesen des Batterystatus

3.6 Android Frontend

Das Android Frontend ist sehr einfach gestrickt. Es besteht aus drei Activities: Der SettingsActivity, welche verwendet wird um die GPS-Tracking Parameter zu konfigurieren, der TrackListActivity welche die Track Liste anzeigt und der TrackDetailActivity - die Detailansicht für die einzelnen Tracks. Sowohl die TrackList als auch das TrackDetail Sheet wurden in Fragments (TrackListFragment und TrackDetailFragment) so implementiert, dass auf einem Tablet nur eine Activity benötigt würde. Auf der linken Seite würde auf einem Tablet die Liste angezeigt und im Zentrum die jeweiligen Detail-Sheets.

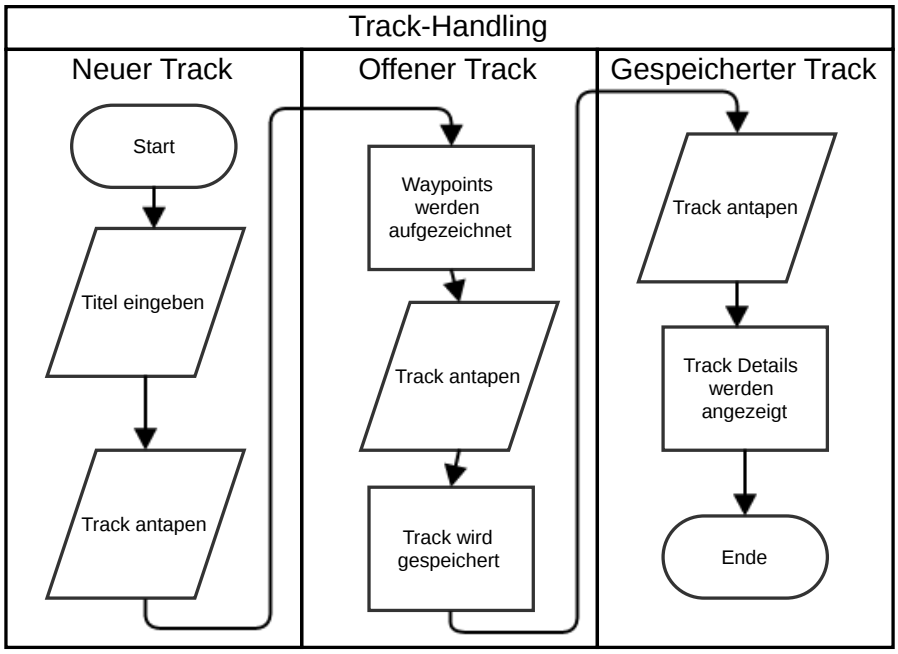

Abbildung 6: Track Lifecycle

Die Benutzerschnittstelle zum Erstellen von Tracks und zum Starten beziehungsweise Beenden des Trackings wurde so einfach wie möglich gehalten. In der Abbildung 6 ist in einem Flowchart der Track Lifecycle ersichtlich. Zum Erstellen eines Tracks, klickt der Benutzer aus der TrackList den Add Track Button. Nach dem Anklicken dieses Buttons wird der Benutzer nach einem Namen für den Track gefragt. Wenn er diesen eingibt und mit Ok bestätigt ist der Track als neuer Track in der Liste ersichtlich. Neue Tracks sind durch ein Play Symbol markiert. Wenn ein neuer Track angeklickt wird, beginnt der Trackingvorgang. Erneutes Anklicken beendet diesen und schreibt den Track in die Datenbank. Wenn gespeicherte Tracks angeklickt werden, öffnet sich das Detail-Sheet auf welchem verschiedene Informationen zum Track angezeigt werden.

3.7 GPX Export

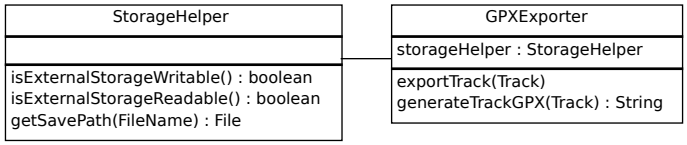

Abbildung 7: Klassendiagramm GPX Export

Das Exportieren von Tracks ins GPX Format wurde in der Klasse GPXExporter umgesetzt. Die Klasse bietet die zwei Methoden generateTrackGPX und exportTrack an. Die erstere nimmt einen Track entgegen und generiert daraus einen GPX XML String nach Spezifikation [17]. Die letztere nimmt einen Track entgegen, findet mithilfe der entsrechenden Helper Klasse einen geeigneten Dateinamen auf Basis des Tracknamens, generiert das GPX XML mithilfe der generateTrackGPX Methode und schreibt dieses in das entsprechende File. Die Files werden im getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS) [6] abgelegt. Auf dieses Verzeichnis kann von einem PC aus zugegriffen werden. Wo genau es liegt, hängt vom verwendeten Gerät ab.

Um Sicherzustellen, dass das externe Directory beschreibbar ist, wurde im Helper die Methode isExternalStorageWritable() implementiert, welche wie folgt prüft, ob das Verzeichnis beschreibbar ist:

```
1 public boolean isExternalStorageWritable() {
2     String state = Environment.getExternalStorageState();
3     if (Environment.MEDIA_MOUNTED.equals(state)) {
4         return true;
5     }
6     return false;
7 }
```

Listing 8: Die isExternalStorageWritable Methode

4 Analyse

4.1 Genauigkeit und Verwendbarkeit der Daten

Die GPS Daten die mit dem verwendeten Gerät (Google Nexus 5) aufgezeichnet wurden, sind - wie für GPS üblich [9] - bis auf wenige Meter genau. Die Genauigkeit hängt dabei vor Allem von der Umgebung ab. In Tunneln oder in engen Häuserschluchten ist der Empfang schlechter als auf einem offenen Feld oder im Wald.

Abbildung 8: Genauigkeitsprobleme - zu wenige Datenpunkte

Probleme durch diese Ungenauigkeit sind vor allem aufgetreten, wenn zu wenig Datenpunkte erfasst wurden. In der Abbildung 8 ist dies gut ersichtlich. Die Sternchen stehen für GPS Datenpunkte. Das erste Sternchen von Rechts liegt ziemlich gut auf der Strasse. Das Zweite ist aufgrund der GPS Ungenauigkeiten und äusseren Einflüssen um einige Meter daneben. Das Dritte ist wieder korrekt auf der Strasse. Zwischen dem Dritten und Vierten bin ich von der Strasse auf einen Waldweg abgebogen. Der vierte Datenpunkt liegt schön auf dem Waldweg, wird der Punkt an dem ich wirklich abgebogen bin (blauer Stern) nicht erfasst. Das Selbe geschieht zwischen dem vierten und fünften Datenpunkt. Diese Probleme lassen sich durch häufigeres Aufzeichnen der Daten minimieren.

Bei einer genügenden Menge Datenpunkte wie in Abbildung 9 existieren immer noch Ungenauigkeiten. Die Route im markierten Bereich verläuft ein bis zwei Meter neben der Strasse. Jedoch wird hier im Gegensatz zur Abbildung 8 der Richtungswechsel problemlos korrekt eingezeichnet.

Man kann die Daten, sofern genügend Datenpunkte vorhanden sind, in der Form in welcher sie derzeit generiert werden durchaus verwenden um den Verlauf einer Fahrradtour festzuhalten. Die im Rahmen dieses Projektes aufgezeichneten Routen sind nicht

Abbildung 9: GPS Ungenauigkeit

befreit von Ungenauigkeiten, jedoch ist überall ersichtlich auf welcher Strasse ich gefahren bin. Um die vorhandenen Ungenauigkeiten zu minimieren, könnte man zum Beispiel einen Map-Matching Filter anwenden wie er von Jason Daniel Martin et al. im Paper Dynamic GPS-position correction for mobile pedestrian navigation and orientation [13] vorgestellt wird. Um die Anzahl Datenpunkte zu verkleinern und um Ausreisser zu eliminieren könnten numerische Verfahren wie zum Beispiel der Ramer-Douglas-Peucker Algorithmus [15][7] verwendet werden.

4.2 Akkuverbrauch

Mit dem Java Android SDK ist es nicht möglich den Akkuverbrauch einer einzelnen Applikation zu messen. Was möglich ist, ist das Messen des aktuellen Akkustandes. In dem erstellen GPS Tracker wurde dies jeweils beim Start und nach Abschluss jedes Tracks getan. Die Auflösung dieser Messwerte sind auf 0.01 oder 1% des Akkus genau.

Die Eingabeparameter des LocationManagers welche einen Einfluss auf den Akkuverbrauch des Trackers haben könnten, sind die im Kapitel 3.1 beschriebenen:

- timeInterval
- minDistance

Wirklich relevant hierbei ist lediglich das timeInterval. In diesen Abständen wird die GPS Location abgerufen. Um den Einfluss dieser Einstellung auf den Akkuverbrauch

zu messen, wurden über rund drei Stunden Tracks mit verschiedenen Zeitintervallen aufgenommen:

Interval (ms)	Runtime (min)	Batteryusage	Points Captured	Batteryusage/Min (Promille)	Batteryusage/Point (Promille)
10000	180.8857666	0.0799999999	863	0.4422680756	0.092699884
5000	182.2862	0.0799999	2180	0.4388697554	0.0366972018
1000	183.6950333	0.0999999	10948	0.5443799879	0.0091340793

Abbildung 10: Statistik Akkuverbrauch

In der Abbildung 10 sind die Resultate dieses Tests ersichtlich. Es hat sich kein markanter Unterschied des Akkuverbrauchs aufgrund dieser Konfigurationsparameter feststellen lässt. Dies deckt sich mit der Stackexchange Antwort vom Stackexchange User Jan:

«The bigger thing is that you will be needing to keep the CPU powered on all the time. And, since you're using the PendingIntent flavor of requestLocationUpdates(), I am guessing that you plan on collecting data for a long time.»

Er beschreibt, dass das Hauptproblem das ständige Arbeiten des Prozessors sei, welches zum Tracken der Location notwendig ist. Gemäss ihm sei es vor allem wichtig, dass man die LocationUpdates wieder ausschaltet wenn man sie nicht mehr braucht. Dies ist der Fall in dem hier hergestellten GPS Tracker.

5 Fazit

Trotz anfäglicher Schwierigkeiten und dem hohen Einarbeitungsaufwand in die Android Entwicklung, ist es gelungen einen Funktionsfähigen GPS Tracker zu entwickeln. Der Export ins GPX Format stellte keine Probleme dar. Die Optimierung des Akkuverbrauchs, welche zu Beginn im Fokus lag, war, wie im Kapitel 4.2 beschrieben, nicht erfolgreich. Jedoch hält sich der Akkuverbrauch in Grenzen, so dass auch Routen vom mehreren Stunden kein Problem darstellen dürften.

Um den GPS Tracker produktiv oder kommerziell nutzen zu können, müssten einige kleine Anpassungen vorgenommen werden. Zum Beispiel müssten Datenbankzugriffe und die Exportfunktion in separate Threads ausgelagert werden, so dass sie das Frontend nicht mehr blockieren. Weiter könnten die gewonnen Daten, zum Beispiel mit den im Kapitel 4.1 beschriebenen Techniken, verfeinert und optmiert werden.

Die Probleme mit der Androidentwicklung widerspiegeln sich auch im Vergleich zwischen der Soll- und der Ist-Planung des Projektes (Kapitel 1.5). Ich musste deshalb vor allem Einschnitte in der Zeit zum Sammeln von Testdaten (dem Velofahren) in kauf nehmen. Auch die Dokumentation der Implementation wurde aus diesem Grund nach hinten verschoben, was sich unter anderem bei der Abgabe der ersten Rohfassung vom 04.06.2014 gezeigt hat.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[1] ArcGIS 9.2. Georeferencing and coordinate systems. `http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=Georeferencing_and_coordinate_systems`.

[2] Leica Geosystems AG. *Einführung in die GPS Vermessung (Global Positioning System)*.

[3] Android Developer API Documentation. Content providers. `http://developer.android.com/guide/topics/providers/content-providers.html`.

[4] Android Developer API Documentation. Locationmanager. `http://developer.android.com/reference/android/location/LocationManager.html`.

[5] Android Developer API Documentation. Monitoring the battery level and charging state. `http://developer.android.com/training/monitoring-device-state/battery-monitoring.html`.

[6] Android Developer API Documentation. Saving files. `http://developer.android.com/training/basics/data-storage/files.html`.

[7] David H Douglas and Peucker Thomas K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Computer Graphics and Image Processing, Vol. 1, No. 3.*, 10(2):112–122, 1973.

[8] Wikimedia Foundation. Create, read, update and delete. `http://en.wikipedia.org/wiki/Create,_read,_update_and_delete`.

[9] Wikimedia Foundation. Global positioning system - genauigkeit der positionsbestimmung. `http://de.wikipedia.org/wiki/Global_Positioning_System#Genauigkeit_der_Positionsbestimmung`.

[10] Wikimedia Foundation. Gps exchange format. `http://en.wikipedia.org/wiki/GPS_Exchange_Format`.

[11] Wikimedia Foundation. Trilateration. `http://de.wikipedia.org/wiki/Trilateration`.

[12] Wikimedia Foundation. World geodetic system 1984. `http://en.wikipedia.org/wiki/World_Geodetic_System_1984`.

[13] Jason Daniel Martin, Jens Krösche, and Susanne Boll. Dynamic gps-position correction for mobile pedestrian navigation and orientation. *Proceedings of the 3rd workshop on positioning, navigation and communication.*

[14] Penstate College of Earth and Mineral Sciences. Geodesy, datums, and coordinate systems. `https://www.e-education.psu.edu/lidar/l3_p3.html`.

[15] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing, Vol. 1, No. 3.*, 1(3):244–256, 1972.

[16] R. W. Sinnott. Virtues of the Haversine. *Sky and Telescope*, 68(2):159+, 1984.

[17] Topografix. Gpx 1.1 schema documentation. `http://www.topografix.com/GPX/1/1/`.