Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D-70569, Stuttgart

Studienarbeit Nr. 2361

Validierung von Straßenkarten mit Hilfe einer Android APP

Penghao Tian

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	M. Sc. Patrick Baier
begonnen am:	23. 12. 2011
beendet am:	09. 07. 2012
CR-Nummer:	C.2.3, C.2.4, C.5

Inhaltsverzeichnis:

Kapitel 1

Einleitung

Das erste Kapitel stellt die Motivation der Studienarbeit vor. Das Hauptziel ist es, Methoden zu entwickeln, durch die eine Straßenkarte nach der Qualitätsanforderung vom Benutzer automatisch validiert wird und dann nach der Geschwindigkeit des mobilen Gerätes eine Klassifizierung bestimmt werden kann. Danach wird ein Überblick der folgenden Kapitel angegeben.

1.1 **Motivation**

Mit der rasanten Entwicklung der Technik von Software und Hardware besitzen Mobiltelefone immer höhere Leistungen und werden mehr und mehr miniaturisiert. Heutzutage dienen sie im alltäglichen Leben nicht nur als einfach, herkömmliche Geräte zur Kommunikation zwischen den Menschen, sondern auch als tragbarer leistungsfähiger Minicomputer. Mit den modernen Mobiltelefonen kann man spielen, fotografieren, im Internet surfen, Emails checken, Dokumente lesen und schreiben, navigieren und so weiter. Man nennt sie deshalb auch „Smartphones“.

Wieso wird ein Mobiltelefon immer „klüger“? Ein wichtiger Grund ist die verstärkte Integration von Sensoren in diesem Mobilgerät. Der Ambient-Light-Sensor (ALS) kann beispielweise die Sichtbarkeit und die Lebensdauer des Akkus von tragbaren Bilderschirmen optimieren [1]. Der Proximity-Sensor erkennt, wie nahe das Display zum Körper ist. Wenn man das Telefon bis zum Ohr bringt, schaltet sich das Display ab, um Energie zu sparen und unerwünschte Eingaben zu vermeiden. Der Accelerometer-Sensor kann die Ausrichtung des Telefons erfassen und die Anzeige ändert sich entsprechend ins Hoch- oder Querformat, so dass es für den Nutzer bequemer ist das Display zu lesen. Es gibt außerdem Sensoren, die

Umweltinformationen geographisches Gebiet ermitteln können. Die dadurch entstehenden Massen der vorhandenen Thermometer sind dazu da, um eine Wärmekarte einer Stadt zu erstellen. Und mit den Messwerten vom GPS nämlich dem „Global Positioning System“ kann man bestimmen, wo z.B. der Benutzer steht, oder eine Route zwischen Positionen A und B erstellen lässt.

Ein vollständig zentralisiertes Modell besteht aus Sensoren zur Datensammlung und den Methoden zur Verarbeitung der gesammelten Sensordaten. Das System wird von Wissenschaftlern gesteuert, die die Sensoren einrichten. Solches ein Modell hat keine Möglichkeit, in eine Stadt zu skalieren, obwohl ein genügender Fond dafür zur Verfügung ist. Die Hauptprobleme sind, dass die Geräte den Wissenschaftlern immer fehlten und die Sensordaten sich nicht gemeinsam teilen lassen. Hierfür wurde das Konzept „Pervasive Computing“ erstmals im Jahr 1988 von Mark Weiser vorgeschlagen. Nach seiner Meinung wird der Computer verschwinden und durch „intelligente Gegenstände“ ersetzt, so dass man irgendwann und irgendwo auf jeden Fall die Daten erfassen und verarbeiten kann. Innerhalb weniger Jahre verbreiten sich die Mobiltelefone schnell ins menschliche Leben. Wegen der Verwendungen der Datentransferfunktion und der eingebauten Sensoren wie GPS oder Thermometer ist es möglich, die Sensordaten kostengünstig zu erfassen. Man bezeichnet die Nutzung von Sensoren allgemein als „Public Sensing“. Sie wurde in unterschiedlichen Dokumenten erklärt und dort hatte sie jeweils andere Namen, wie beispielsweise „Public Urban Sensing“ [2], „Mobile Phone Sensing“ [3] oder „Mobiscopes“ [4].

Eine wichtige Anwendung der aus verschiedenen GPS-Sensoren erhaltenen geographischen Daten ist die Erstellung einer Landkarte. Eine Methode wie OpenStreetMap [5] ist manuell. Ein GPS Trace ist eine Folge von genauen Positionen. Jeder GPS-Trace ist deshalb eine polygonale Linie mit einem einzigen Start- und Endpunkt. Freiwillige erzeugen nützliche Karte aus den gesammelten GPS Traces und integrieren sie in eine globale Karte. Im Gegensatz dazu wird empfohlen, durch Folgerung aus den bekannten GPS Traces die Straßenkarten automatisch zu generieren (z.B. [6], [7]). Der Nachteil beider Ansätze ist, dass die Qualität der resultierenden Straßenkarte schwer auszuwerten ist. Das heißt, die Karte unzuverlässig ist. Zum Beispiel hängt die Genauigkeit der Wegpunkte der Karte stark von der Genauigkeit der GPS-Sensoren ab, die benutzt werden, um GPS Traces zu sammeln. Darüber hinaus könnte die Geometrie einer Straßenkarte vielleicht in der

Zukunft rekonstruiert werden. Aus diesem Grund kann diese Karte nicht in kritischen Situationen angewendet werden.

Um die Gültigkeit und die Zuverlässigkeit einer gegebenen Straßenkarte sicherzustellen braucht man einen Mechanismus, der die Karte automatisch korrigieren und validieren kann. Das ist die Forschungsfrage dieser Studienarbeit und wird im Kapitel 3 genau erläutert.

1.2 Ziel der Studienarbeit

In dieser Studienarbeit untersucht man eine Methode, die sich damit befasst wie diese GPS-Sensoren genutzt werden können, um eine gegebene Straßenkarte automatische hinsichtlich ihrer Genauigkeit zu analysieren. Hierbei wurden in einer ersten Arbeit bereits theoretische Ergebnisse erzielt, die zeigen, dass mit Hilfe von GPS Wegpunkten eine vorhandene Straßenkarte effizient validiert werden kann.

Im ersten Schritt der Studienarbeit sollen nun bisher theoretisch erzielte Ergebnisse, mit Hilfe einer Android-Implementierung auch praktisch belegt werden. Dazu soll eine Android-Applikation entwickelt werden, die automatisch gesammelte Wegpunkte auf Basis der bereits vorhandenen Arbeit nutzt, um automatisch eine gegebene Straßenkarte zu validieren.

Das bisher entwickelte Verfahren berücksichtigt noch nicht die Geschwindigkeit der Mobilgeräte, welche die für die Validierung nötigen GPS Wegpunkte aufzeichnen. Da diese einen entscheidenden Einfluss auf die Kartenvalidierung hat, soll im zweiten Schritt der Studienarbeit ein Verfahren entwickelt werden, welches die Geschwindigkeit der Mobilgeräte anhand der GPS Daten klassifiziert. Mögliche Kategorien wären zum Beispiel: Fußgänger, Radfahrer oder Autofahrer. Auf Basis dieser Klassifizierung soll der vorhandene Algorithmus zur Kartenvalidierung erweitert werden, mit dem Ziel die Effizienz des Systems zu steigern.

1.3 Aufbau der Studienarbeit

Die restlichen Teile der Studienarbeit sind wie folgt organisiert.

Kapitel 2: „Verwandte Arbeiten“

Hier werden die Algorithmen zur Erstellung einer Landkarte und etwas Informationen über OpenStreetMap, Google Maps und Android erklärt, um die Studienarbeit gut verstehen zu können.

Kapitel 3: „Konzept“

Hier wird zuerst das System Model eingeführt. Dann werden der grundlegende Entwurf über die Validierung und Korrektur einer Straßenkarte und der Algorithmus zur Klassifizierung von Geschwindigkeit des Mobilgerätes konkret dargestellt.

Kapitel 4: „Implementierung“

Die Entwürfe, die auf dem Konzept im Kapitel 3 basieren, werden hier detailliert implementiert.

Kapitel 5: „Evaluation“

Die Implementierungen der Methoden und ihre Ergebnisse werden hier durch einige Trace Proben überprüft, ob sie richtig funktionieren.

Kapitel 6: „Zusammenfassung“

Die Ausarbeitung wird hier zusammengefasst.

Kapitel 2

Verwandte Arbeiten

Zuerst müssen wir uns darauf konzentrieren, wie eine Straßenkarte automatisch erstellt werden kann. Da alle Methoden zur Validierung und Klassifizierung auf einer gegebenen Straßenkarte durchgeführt werden, werden hier zwei schon verbreitete Karten vorgestellt. Dann wird die Android Plattform angegeben, auf der die GPS Traces gesammelt werden.

2.1 Automatische Erstellung einer Straßenkarte

Die erste Methode zur automatischen Konstruktion einer Straßenkarte wurde von Morris et al [8] vorgeschlagen. Man muss anfangs aus den gesammelten GPS Traces einen primären Graph konstruieren. Jeder GPS Trace wird als einen einzigen Pfad betrachtet. Der Graph wird durch alle Schnittpunkte aller vorhandenen GPS Traces dargestellt. Ein zweidimensionaler Schnittpunktetest zwischen allen einzelnen Liniensegmenten kann in der Zeit O (n log n + k log n) erreicht werden [9], wobei n die Anzahl der Segmente und k die Anzahl der Schnittpunkte ist. Sobald die Schnittpunkte gefunden wurden, werden die Traces dann dadurch in die Segmente untergeteilt, welche die Kanten des Graphs bilden. Um die Kanten zu verschmelzen, die so nahe sind, dass sie dupliziert einen denselben Path darstellen, soll man weiter eine Reihe der graphischen Reduktionsoperationen durchführen. Diese Kanten können (wie in der Abbildung 1 angezeigt) gemittelt und zuletzt durch eine eindeutige Darstellung ersetzt werden.

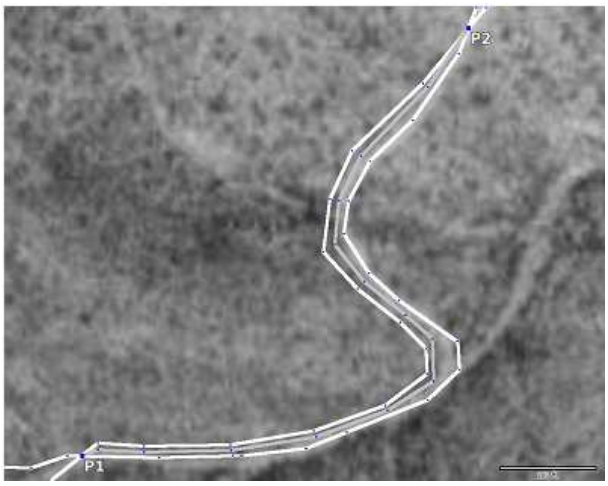


Abbildung1: parallele Kanten-Reduktion. Die zwei äußeren Polylinien zwischen P_1 und P_2 sind die ursprünglichen Kanten. Bei dem Reduktionsschritt wird die zwei Kanten durch die mittlere Linie ersetzt, die in der Mitte liegt [8].

Bisher gibt es noch viele andere Algorithmen zur Kartengeneration. Jeder kann theoretisch einige „gleichen Path beschreibende“ GPS Traces in eine eindeutige Darstellung einer Straßenkarte fusionieren z.B. [6], [7], [10], [11]. Im Dokument [7] stellten Brünstrup et al. einen inkrementellen Ansatz vor, der mit einem einzigen GPS Trace beginnt und nacheinander die ähnlichen Traces mit dem schon vorhandenen Trace verschmilzt, um die Geometrie einer Straße zu ermessen.

Neben diesen allgemeinen Algorithmen zur Kartenerstellung werden hier die komplexeren Probleme mehr beobachtet, wie man aus den überlappenden GPS Traces die Schnittpunkten ableiten [12] oder wie man durch Analysieren der Positionen und Kommunikationsgeschichte die Hindernisse erfassen kann [13]. Im Unterschied zu den oben genannten Algorithmen zur Folgerung der Struktur der Straßen sind die Ansätze existent, die den Aufbau der Straßen im Voraus bereits wissen und sich nur auf seinen gegenwärtigen Zustand konzentrieren. Das Pothole Patrol System [14] versucht, die Anomalien auf den Straßen dadurch zu erkennen, dass die Daten über Beschleunigung von Autos analysiert werden. Das Nericell System [15] hat eine ähnliche Methode, aber zusätzlich analysiert es noch die Daten von Audio-Sensoren.

Der Ansatz dieser Studienarbeit ist in Bezug auf die Kartegeneration von Rogers et al. [16]. Die GPS Traces werden genutzt, um eine bereits bestehende Straßenkarte zu

verfeinern und schließlich versucht man, die Art der gegebenen Straßen zu bestimmen. Dieser Ansatz ist gut verbessert, weil er die expliziten Metriken über Qualität einrichtet, so dass es kontrolliert wird, ob die Genauigkeit einer Straßenkarte die Qualitätsanforderungen von den Benutzern erfüllen kann.

2.2 OpenStreetMap und Google Maps

In dieser Studienarbeit wird keine neue Straßenkarte erzeugt, sondern es wird eine bereits vorhandene Karte genutzt um geographische Daten über einer Straße zu lesen, die einem vom mobilen Gerät gesammelten GPS Trace entspricht, um beide zusammenführen und weiter die neue Geometrie der Straße mit den Qualitätsmetriken vergleichen zu können. Es gibt zwei bekannte kostenlose Weltkarten OpenStreetMap und Google Maps.

Die geographischen Informationen werden derzeitig immer wichtiger aber sie sind selten kostenlos erhältlich. Man muss eine Lizenz für Kartenmaterial vom Besitzer kaufen und unter Umständen ist sie sehr teuer. Wer ein Navigationsgerät nutzen will, muss das digitale Kartenmaterial für z.B. Europa bezahlen. Oftmals ist das Material nach dem Kauf unvollständig oder wird in Kürze veraltet sein.

Abbildung 2: Logos von OSM und Google Maps

Das OpenStreetMap (OSM) Projekt versucht, alle nutzbaren Geodaten zu sammeln und daraus eine freie Weltkarte zu erstellen. Es wurde im Jahr 2004 von Steve Coast aufgestellt. Der Vorgang zur Erstellung einer Karte ist ähnlich wie der Wikipedia. Freiwillige tragen die Geodaten zusammen, die z.B. von GPS Empfängern gesammelt oder aus vorhandenen digitalen Karten übertragen werden. Nach dem Hochladen werden die rohen Daten weiter ediert und die zusätzlichen Informationen darüber werden eingefügt. Weil die Karte von vielen Freiwilligen zusammen erzeugt wird, gibt es kein Risiko über das Eigentum. Man kann mit einer freien Lizenz die

Daten aus der Karte nutzen oder die Karte in Homepages einbetten [22] [23].

Es ist bereits üblich Kartendienste im Internet zu nutzen. Mit diesen Diensten kann man schnell mal gucken, wie man vom Wohnort zum Restaurant kommt, wo sich der Benutzer befindet, usw. Der bevorzugte Anbieter des Dienstes ist Google Maps natürlich. Google Maps stellt jeden Ort dar und bietet fast alle Informationen über eine Straße inklusive Hausnummer, Postleitzahl, Ort, Satellitenbilder, dazu Wegbeschreibung und so weiter. Google Maps Dienst wurde im Jahr 2005 von Google Inc. gestartet. Mit ihm kann man ein bestimmtes Objekt auf der Weltkarte suchen, um seine Informationen genau anzuzeigen. Er bietet auch für viele Länder einen Routenplaner an. Und seit 2008 sind für manche Länder Echtzeit-Verkehrsinformationen integriert. Nun kann man die Karte abspeichern, freigeben und neue Overlays erstellen, um eigene Google Maps darzustellen. Google Maps ist kostenlos aber nicht frei. Das Material über die Karte unterliegt den Lizenzbestimmungen von Google [24].

Die zwei oben genannten Karten sind heute für Mobiltelefone oder PDAs nutzbar. Auf vielen mobilen Endgeräten ist Google Maps bereits vorinstalliert, so dass man den Kartendienst mobil nutzen kann. Es gibt viele Versionen für z.B. Java, Symbian, und Android.

2.3 Android

Da ein Erfassen der GPS Traces auf der Android implementiert wird, wird hier etwas für die Entwicklungsplattform eingeführt.

Android ist sowohl ein Betriebssystem als auch eine Software-Plattform für mobile Geräte wie Smartphones, Mobiltelefone und Tablets. Es wird von der Open Handset Alliance entwickelt, um freie Software quelloffen entwickeln zu können [25]. Seine Basis ist der Linux Kernel. Android nutzt Linux zur Verwaltung der Speicher, Prozesse, Netzwerke und anderen Operationen. Auf der höchsten Ebene liegen die Applikationen. Der Anwender kann nur diese Applikationen sehen und bedienen. Jede Benutzerschnittstelle wird von Aktivität Class dargestellt. Jede Aktivität hat ihren eigenen Lebenszyklus (Abbildung 3). Eine Applikation besteht aus einer oder mehr Aktivitäten.

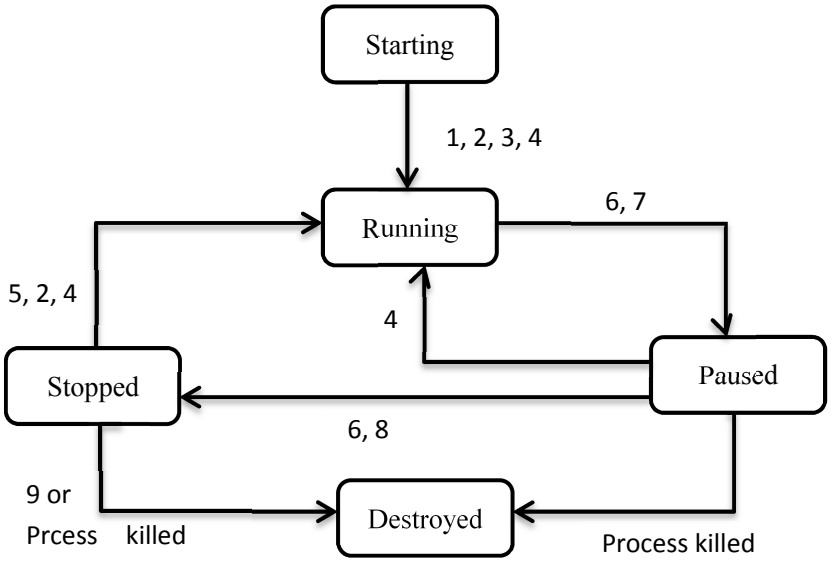


Abbildung 3: Lebenszyklus einer Android Aktivität [26]

Innerhalb der Lebensdauer kann sich eine Aktivität in einer dieser Zustände befinden. Die Änderung der Zustände wird durch die Methode *onXXX( )* gesteuert. Man kann die Methoden überschreiben und Android kann sie am richtigen Zeitpunkt aufrufen [26].

1. *onCreate(Bundle)*. Beim ersten Start einer Aktivität wird sie aufgerufen, um einmalige Initialisierungen, wie das Erstellen einer Benutzeroberfläche, auszuführen. Der Parameter ist entweder null oder eine Zustandsinformation, die vorher durch die *onSaveInstanceState( )* Methode gespeichert wurde.

2. *onStart( )*. Sie zeigt eine Aktivität, die dem Benutzer angezeigt wird.

3. *onRestoreInstanceState(Bundle)*. Sie wird aufgerufen, wenn die Aktivität durch einen zuvor von *onSaveInstanceState( )* gespeichertem Zustand neu initialisiert wird. Die Default Implementierung setzt die Benutzeroberfläche zurück.

4. *onResume( )*. Die wird aufgerufen, wenn eine Interaktion zwischen Aktivität und Benutzer beginnt.

5. *onRestart( )*. Der Aufruf dieser Methode bedeutet eine erneute Anzeige der Aktivität von einem gestoppten Zustand.

6. *onSaveInstanceState(Bundle)*. Android ruft diese Methode auf, um den Zustand

für jede Instanz zu speichern.

7. *onPause( )*. Wenn eine Aktivität zur Arbeit im Hintergrund geht, läuft *onPause( )*, weil eine andere Aktivität vor dem Benutzer arbeitet.
8. *onStop( )*. Wenn eine Aktivität nicht mehr benötigt ist, kann man *onStop( )* aufrufen.
9. *onDestroy( )*. Vor der Zerstörung einer Aktivität läuft sie. Wenn der Speicher knapp ist, kann *onDestroy( )* nie aufgerufen werden.

Neben der Verwaltung des Lebenszyklus des Programms bietet der Android Rahmen die Bausteine wie Activities, Intents, Services, Content Providers, usw. zur Erstellung von den Applikationen. Diese werden hier nicht mehr vorgestellt.

Kapitel 3

Konzept

In diesem Kapitel wird vor Allem unserer System Model und die Annahmen des Systems vorgestellt. Um die Idee der Korrektur und Validierung einer Straßenkarte gut erklären zu können soll dann die Qualität Metriken eingeführt werden, die bereits in dem Abschnitt 2.1 kurz erwähnt wurde, bevor wir erläutern, wie das System mit einer vorhandenen Validierungsabfrage initialisiert werde und wie GPS Traces zur Verfeinerung und Validierung einer Straßenkarte weiter bearbeitet werden.

3.1 System Model

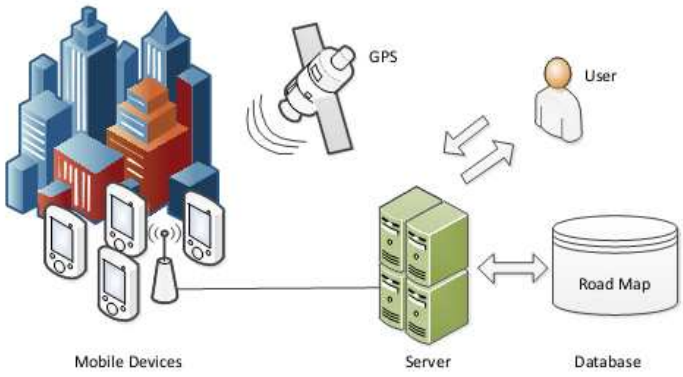


Abbildung 4: System Model [17]

Das System besteht aus einem zentralen Server und einigen mobilen Geräten. Ein Mobilgerät hat einen eingebauten GPS-Sensor und kommuniziert mit dem zentralen Server über ein zellulares Mobilfunknetz wie UMTS (sehen Abbildung 4). Es muss zuerst angenommen werden, dass der Server diese zum Erfassen der Sensordaten

verfügbaren mobilen Geräten weiß. Durch einen Registrierungsmechanismus in [18] kann dies erreicht werden. Zweitens hat der Server keinen Einfluss auf die Bewegungsrichtung und Geschwindigkeit der Geräte und die Bewegung der Geräte muss die zugrundeliegende Struktur der Straßen befolgen. Weiter hat der zentrale Server unbegrenzte Rechenleistung und Energieversorgung und die Energie eines mobilen Gerätes wird durch seine Batterie beschränkt. Deshalb haben die zu verwendenden Operationen wie GPS Positionsbestimmung und die Kommunikation zwischen Server und Clients entscheidende Bedeutung und müssen minimiert werden.

Die Positionsinformationen der bewegten als auch stationären Objekte haben in der Regel eine Ungenauigkeit. Aufgrund physikalischer Beschränkung und Messfehler der Messhardware sind die durch GPS Empfänger erfassten Positionsdaten nur annähernd zu den tatsächlichen Positionen des jeweiligen Sensors. Um die Kommunikationskosten zu reduzieren lassen sich die Positionsinformationen durch Update Protokolle für Übertragung von Sensoren zur entfernten Datenbanken oder lokalen Diensten verringern. Die Interpolationsberechnung zwischen aufeinander folgenden Positionspaaren kann weiter die Ungenauigkeit verstärken. Obwohl eine Fusion der Positionsdaten auf der gleichen Straße die Genauigkeit verbessern kann, kann diese Genauigkeit nicht beseitigt werden.

Ein mobiles Gerät erhält eine Koordinate und einen Fehlerwert für jede GPS Position. Jede Koordinate enthält zwei Werte nämlich einen Latitude Wert und einen Longitude Wert. Der Fehler Wert beschreibt die Genauigkeit von einer Koordinate. Im Dokument [19] wurde der Fehler Wert durch die Standardabweichung σ der erhaltenen Positionsdaten dargestellt. Unter der Berücksichtigung der relativen Geometrie der Satelliten für GPS-Fix kann „Horizontal Dilution of Precision“ (HDOP) abgeleitet werden. Der Wert von HDOP beschreibt die Qualität der Positionsbestimmung und liegt theoretisch von 0 bis 50. Je kleiner der Wert, desto genauer die Positionsbestimmung. Ein guter Wert meint den maximalen Wert bis 2,5. Für Navigation ist der Wert bis 8 noch akzeptabel. Mit der Hilfe dieses Wertes und „ User Equivalent Range Error“ (σ_{UERE}) kann man die Standardabweichung σ berechnen [20]:

$$\sigma = HDOP * \sigma_{UERE} \quad (1)$$

Daher bedeutet ein kleinerer Wert von σ eine höhere Wahrscheinlichkeit darüber, dass die wahre Position des mobilen Gerätes in der Nähe von der gewonnenen Koordinate liegt. Eine Annahme ist, dass alle GPS-Sensoren die Werte von HDOP und σ_{UERE} bekommen und deswegen die Standardabweichung σ einer GPS-Position berechnen können. So nennen wir den Wert auch als p_{gps}. σ.

Die schon überprüfte Straßenkarte soll in einer Datenbank gespeichert und durch den Server zugegriffen werden. In der Annahme kennen die mobilen Geräte die Karte, sonst muss der Server die Karte an die Endgeräte senden. Die Straßenkarte ist ein Graph über Straße, der aus Kanten und Knoten besteht. Die Knoten stellt die Schnittpunkte der Wege und die Endpunkte einer Straße dar. Die Kanten als die Straßen durch die Knoten miteinander zusammengeknüpft sind. Jede Kante ist eine Reihe verbundener Punkte, die die Geometrie einer Straße formen (wie in Abbildung 5a). Ein Punkt auf der Straße hat einen zugewiesenen Fehler: e, der die Genauigkeit eines Wegpunktes analog zu dem Fehler Wert einer GPS Position p_{gps}. σ. Deshalb ist die Geometrie, die die Wegpunkte mit geringen Fehlerwerten bilden, relativ annähernder zu der realen Geometrie der Straße.

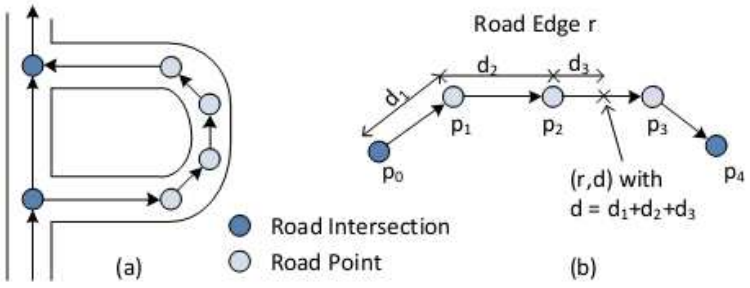


Abbildung 5: ein Beispiel eines Straßengraphs [17]

Jede Kante hat eine eindeutige Nummer und der erste Punkt ist der Ausgangspunkt. Wir können einen Punkt (r, d) bestimmen, wobei der Punkt auf die Straße r liegt und d die Abstand von dem Punkt zum Ausgangspunkt ist (wie in Abbildung 5b). Ein Segment einer Kante mit Startpunkt p_i und Endpunkt p_j (i < j) kann durch $<p_i, p_j>$ = $(p_i, p_{i+1}, \ldots, p_{j-1}, p_j,)$ angegeben werden.

3.2 Qualität Metriken

Außer, dass die Geometrie einer gegebenen Straßenkarte korrigiert werden soll, soll das System dem Benutzer eine Gruppe von Segmenten liefern, die in Bezug auf die Genauigkeit validiert werden. Somit müssen wir eine Methode bestimmen, um die Qualität eines vorhandenen Straßensegments zu messen und dann entscheiden, ob das Segment als ein Validiertes betrachtet werden kann. Diese Schätzungen sind sehr subjektiv. Deshalb werden die Qualitätsanforderungen nachfolgend vorgestellt, die vom Benutzer angegeben werden und in der Validierungsabfrage eingeschlossen sind.

Die Ausgaben der Erfassungen der mobilen Geräte sind GPS Traces. Sie werden anschließend am zentralen Server verschickt und dort verarbeitet. Diese GPS Traces nutzt man weiter, um mit den zusätzlichen Wegpunkten einen vorhandenen Straßengraph zu bereichern. Der Fehler Wert p.e von einem neuen Punkt kann man aus dem Positionsfehler p_{gps}. σ berechnen. Durch die Auswertung der räumlichen Verteilung und die Genauigkeit der Wegpunkte eines Straßensegments kann man entscheiden, ob dieses Segment als ein validiertes Segment gesehen wird. Nun führen wir daher zwei vom Benutzer vorgegebene Parameter ein, die eine maximale Distanz d_{max} und einen maximalen Fehler e_{max} einer Reihe von aufeinander folgenden Punkten auf den Straßen definieren.

Sei $<p_i, p_j>$ ein Segment einer Straße auf einem Straßengraph. Wenn es ein validiertes Segment ist, muss es die beiden folgenden Kriterien erfüllen, wobei $d(p_x,p_y)$ die Länge der Kanten zwischen Wegpunkten p_x und p_y:

$$\forall k \in \{i, \dots, j-1\} : d(p_k, p_{k+1}) \leq d_{max} \quad (2)$$

$$\forall k \in \{i, \dots, j\} : p_k.e \leq e_{max} \quad (3)$$

Beide Anforderungen sind über die Genauigkeit der Form des Straßensegments eingestellt. Die Gleichung (2) sorgt dafür, dass der maximale Abstand zwischen jedem Paar von aufeinanderfolgenden Wegpunkten nicht größer als den vorgegebenen Schwellenwert d_{max} ist. Offensichtlich kann eine komplexe Form wie eine Kurve nur genau angenähert werden, wenn die Wegpunkte genug nah liegen (Abbildung 6). Die Gleichung (3) stellt sicher, dass der Fehler Wert aller Wegpunkte

im Segment $<p_i , p_j>$ die maximale Fehlereinschränkung e_{max} nicht überschreiten kann. So wird sichergestellt, dass die Wegpunkte in Hinblick auf die Genauigkeit der Position hinreichend abgebildet werden.

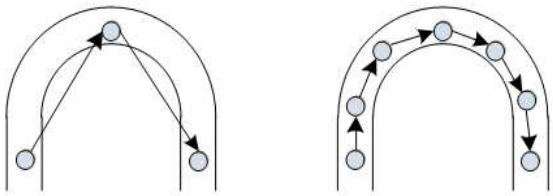


Abbildung 6: Annäherung einer Kurve [17]

3.3 Initialisierung des Systems

Das System beginnt mit einer Validierungsabfrage vom Benutzer. Diese Abfrage enthält die oben definierten Qualität Metriken und die Straßenkarte soll dadurch validiert werden. Wir initialisieren alle Wegpunkte auf dem Straßengraph mit einem anfänglichen Fehlerwert. Falls keine Genauigkeitswerte für die ursprüngliche Karte existiert, richten wir ihre Wegpunkte mit einem schlimmsten Fehlerwert ein. Andernfalls kann das System mit bereits bekannten Werten über Genauigkeit für diese Punkte beginnen.

Am Anfang sendet der zentrale Server eine Nachricht zur Initialisierung an allen bekannten mobilen Geräten, die sich auch in diesem Bereich befinden. Nach dem Empfang einer solchen Nachricht, zeichnen alle teilnehmenden Geräte kontinuierlich die GPS-Positionen pro t_{gps} Sekunde auf. Wenn ein mobiles Gerät eine vordefinierte Anzahl der Positionen (z.B. 100 GPS Positionen) erfasst, schickt es dem Server diese GPS Traces.

3.4 Verarbeitung der GPS Traces

Nachdem der Server die GPS Traces eines mobilen Gerätes erhalten hat, vergleicht es sie mit dem Straßengraph. Es wird jetzt angenommen, dass jeder GPS Trace eine entsprechende Straße im Graph finden kann. Anderenfalls könnte der Trace entweder

auf einer Straße liegen, die zum keinen Teil der gegebenen Straßenkarte gehört, oder das mobile Gerät sich derzeit nicht auf einer bekannten Straße befindet. Wir konzentrieren uns in der Studienarbeit nur auf die Validierung einer Straßenkarte, so dass die nicht zu einem Segment einer existierenden Straße zugeordneten GPS Traces herausgefiltert werden müssen. Trotzdem sind die hier herausgefilterten GPS Traces nutzbar für manche Algorithmen zur Kartengenerationen. Damit können die zusätzlichen Straßen in der Karte eingefügt werden.

Mit den Fusionsalgorithmen (z.B. [8], [21]) kann ein GPS Trace mit einer existierenden Straße zusammengeführt werden. Hier wird der Roth's Ansatz genau skizziert. Im Dokument [21] nutzte Roth ein verhältnismäßiges Modell zur Verschmelzung zweier entsprechender Traces in einen eindeutigen Path.

Seien zwei Segmente T_1 und T_2 schon gegeben. Sie stellen einen gleichen Path dar. Ein Messungswert ist ein Tripel (x, y, var), wobei (x, y) die Koordinate und var die Varianz der entsprechenden Gauß-Verteilung in Bezug auf den Messungswert ist. Man berechnet einen Tripelwert aus zwei Tripelwerten, der als ein Eingabewert für weitere Fusionsberechnung werden kann. Ein Beispiel dafür befindet sich in der Abbildung 7.

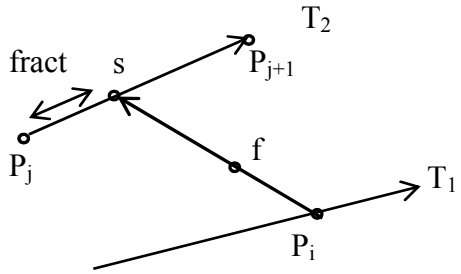


Abbildung 7: Beispiel für Fusionsberechnung [17]

P_i ist eine Position auf T_1. P_j und P_{j+1} sind Positionen auf T_2. s ist der Schnittpunkt einer geraden Linie von P_i zum T_2 und die Länge der Linie zwischen s und P_i ist die minimale Distanz vom P_i zum Trace T_2. Eine Position $f = (x_f, y_f, var_f)$ wird nach einer Fusionsberechnung neu erstellt. Aus den gegebenen Tripelwerten von P_j und P_{j+1}und der minimalen Distanz von P_i zum T_2, kann man das Tripel $s = (x_s, y_s, var_s)$ auf dem Segment $<P_j, P_{j+1}>$ ausrechnen. Der Wert von var_s muss aber doch berechnet werden.

$$var_s = var_{P_j} * (1 - fract) + var_{P_{j+1}} * fract \quad (4)$$

wobei der Anteil $fract = | < P_j,\ s > / < P_j, P_{j+1} > |$.

Die Formel (4) ist die lineare Interpolation zwischen beiden Varianzen von P_j und P_{j+1}. Aus (x_s, y_s, var_s) und (x_{Pi}, y_{Pi}, var_{Pi}) wird die kombinierte Position f durch Formel (5) ausrechnet.

$$f = \begin{pmatrix} x_f \\ y_f \\ var_f \end{pmatrix} = \begin{pmatrix} (x_s * var_{P_i} + x_{P_i} * var_s)/(var_{P_i} + var_s) \\ (y_s * var_{P_i} + y_{P_i} * var_s)/(var_{P_i} + var_s) \\ (var_{P_i} * var_s)/(var_{P_i} + var_s) \end{pmatrix} \quad (5)$$

Offensichtlich ist die Varianz vom neuen Punkt f immer kleiner als die niedrigste Varianz der beiden Eingabepositionen. Hieraus lassen sich die Varianzen der Punkte immer verbessern, wenn die Leute mehrmals auf dem Weg gehen.

Darunterliegend ist der Ansatz der Ausarbeitung zur Verschmelzung einer Straße mit dem entsprechenden GPS Trace. Er geht von dem oben vorgestellten Ansatz von Roth aus und lässt sich verändern.

Falls eine GPS Position p_{gps} gegeben ist, kann man einen entsprechenden Wegpunkt p_m auf der gegebenen Straße finden. Beim Algorithmus von Roth basiert die Bestimmung des Punktes p_m auf ein Histogramm und ist sehr komplex. Hier lässt sich dieser Schritt vereinfachen, um ihn leicht implementieren zu können. Man berechnet alle Distanzen zwischen GPS Position p_{gps} auf dem GPS Trace und allen Punkten (p_1, … p_m, …p_n) auf der dem GPS Trace entsprechenden Straße. Wenn ein Punkt p_m ausgewählt werden kann, besitzt er den minimalen Abstand dazwischen. Hier kann ein Schwellenwert für diese minimale Distanz eingesetzt werden, um die Werte auszuschließen, die zu groß sind, damit sich der Umfang zur p_m Bestimmung wahrscheinlich verringern kann. Schließlich kann man für jeden Punkt auf dem GPS Trace einen entsprechenden Punkt auf der Straße bestimmen.

Der Fehlerwert $p_m.e$ des Punktes kann am Anfang vom Benutzer eingerichtet werden (z.B. $p_m.e$ =1 Meter).

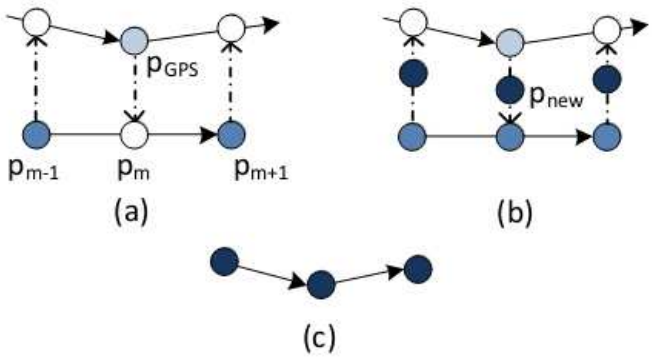


Abbildung 8: die Verarbeitung des GPS Traces [17]

Die GPS Position p_{gps} und den Wegpunkt p_m nutzt man weiter zur Korrektur der Straßengeometrie dadurch, dass ein neuer Wegpunkt p_{new} berechnet wird (Abbildung 8b). Um die Koordinate dieses neuen Punktes zu bekommen, wird eine gewichtete Interpolation durchgeführt, die auf die Standardabweichung $p_{gps}.\sigma$ und den Fehlerwert $p_m.e$:

$$p_{new}.x = \frac{p_m.x * p_{gps}.\sigma + p_{gps}.x * p_m.e}{(p_m.e + p_{gps}.\sigma)} \tag{6}$$

$$p_{new}.y = \frac{p_m.y * p_{gps}.\sigma + p_{gps}.y * p_m.e}{(p_m.e + p_{gps}.\sigma)} \tag{7}$$

Darüber hinaus ist der Fehler Wert des neuen Punktes durch Verwendung von Bayes bedingten Wahrscheinlichkeiten auf die Gauß Verteilung ausgerechnet [16]:

$$p_{new}.e = \frac{p_m.e * p_{gps}.\sigma}{(p_m.e + p_{gps}.\sigma)} \tag{8}$$

Aus der Formel (8) kann man leicht herauslesen, dass im Vergleich zu $p_m.e$ der Wert von $p_{new}.e$ abnimmt. Deshalb wird die Glaubwürdigkeit der neuer Position höher. Wir entfernen p_m auf der Straße und fügen p_{new} an diesem entsprechenden Position ein (Abbildung 8c). Auf diese Weise wird ein GPS Trace mit seinem entsprechenden Straßensegment verschmelzt und die Fehlerwerte aller Punkte auf dem neuen erzeugten Segment verkleinert sich.

Nach der Verschmelzung überprüft der Server alle geänderten Straßensegmente, ob sie die im Abschnitt 3.2 definierten Qualitätskriterien erfüllen. Die Überprüfung

beginnt mit Wegpunkt p_0 und wird mit den nachfolgenden Punkten weiter durchgeführt, bis ein Punkt p_i die Bedingungen verletzt. Der Server markiert dann das Segment $<p_0, p_{i-1}>$ als validiertes Segment und der neue Überprüfungsvorgang arbeitet vom neuen Startpunkt p_i weiter. Das kann in der linearen Zeit fertiggestellt werden. Zum Schluss kann der Server ein Paar validierte Segmente für jede Straße berechnen.

3.5 Klassifizierung nach Geschwindigkeit

Bisher wurde die Geschwindigkeit der Mobilgräte nicht berücksichtigt. In diesem Abschnitt wird eine Methode entwickelt, um die Geschwindigkeit anhand der GPS Daten zu klassifizieren.

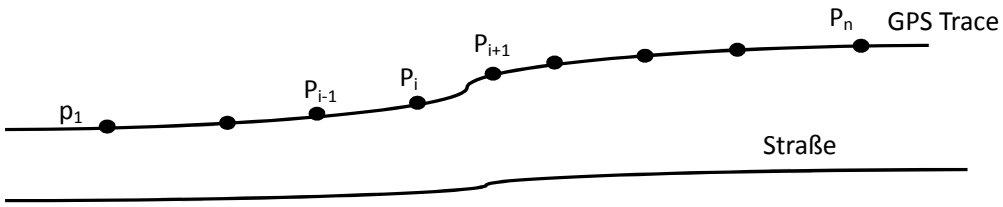


Abbildung 9: Geschwindigkeit berechnen

In der Abbildung 9 steht ein GPS Trace mit seinen erfassten GPS Positionen p_1, …, p_n, wobei n die Anzahl aller GPS Punkte ist.

Am einfachsten berechnet man eine durchschnittliche Geschwindigkeit zwischen Positionen p_1 und p_n durch die Formel (9). Dann vergleicht man den Wert mit den Schwellenwerten (10) (11) (12), um eine Klassifikation der Geschwindigkeit zu entscheiden.

$$v_{i,j} = \frac{d_{i,j}}{t_{i,j}} \tag{9}$$

wobei $d_{i,j}$ die Distanz zwischen Positionen i und j, $t_{i,j}$ der Differenz zwischen Positionen i und j, und $v_{i,j}$ die Durchschnittsgeschwindigkeit zwischen Positionen i und j ist.

Hier werden nur drei Klassifikationen definiert. Die Schwellenwerte dafür können

wie folgt eingerichtet werden [27]. Die Einheit ist Meter pro Sekunde.

Fußgänger: $0 \leq v_{i,j} < 2$ (10)

Radfahrer: $2 \leq v_{i,j} < 5$ (11)

Autofahrer: $5 \leq v_{i,j}$ (12)

Aber nur durch einfache Berechnung kann man wahrscheinlich keine richtige Klassifikation bekommen. Es sind noch zu viele Störungen vorhanden.

- Man ändert oft seine Transportmethoden. Man geht zur Bushaltestelle und steigt sofort ein. Das heißt, die Geschwindigkeit ändert sich oft.
- Die Geschwindigkeit hängt von Verkehrssituation oder Wetter ab. Im Stau würde die mittlere Geschwindigkeit der Fahrt so langsam sein wie beim Radfahren.
- Die Geschwindigkeiten beim Gehen, Laufen und langsamen Radfahren haben keine deutlichen Schwellenwerte.

Das bedeutet, dass die eingerichteten Schwellenwerte nicht verlässliche Kriterien sind und die Geschwindigkeit nicht als ein durchschnittlicher Wert einfach zu berücksichtigen sind. Um die Richtigkeit der Entscheidung zu verbessern, wird hier ein verhältnismäßiges Modell genutzt.

Man berechnet die Geschwindigkeit von je zwei aufeinander folgenden Positionen. Diese wird mit den Schwellenwerten verglichen und die Anzahl für jede Klassifikation wird aufgezeichnet. Man kann somit eine Verteilung aller Geschwindigkeiten erhalten. Wenn die meisten Geschwindigkeiten (z.B. 70%) ein Kriterium erfüllen, kann man eine Klassifikation entsprechend bestimmen. Die Glaubwürdigkeit der Entscheidung erhöht sich stark.

Die Abbildung 10 zeigt eine Verteilung als ein Beispiel. 23 von 30 (77%) Geschwindigkeitswerten befinden sich in der Zone der Fußgänger-Geschwindigkeiten von der Formel (10). Man kann daraus sagen, dass der Träger des mobilen Gerätes ein Fußgänger ist.

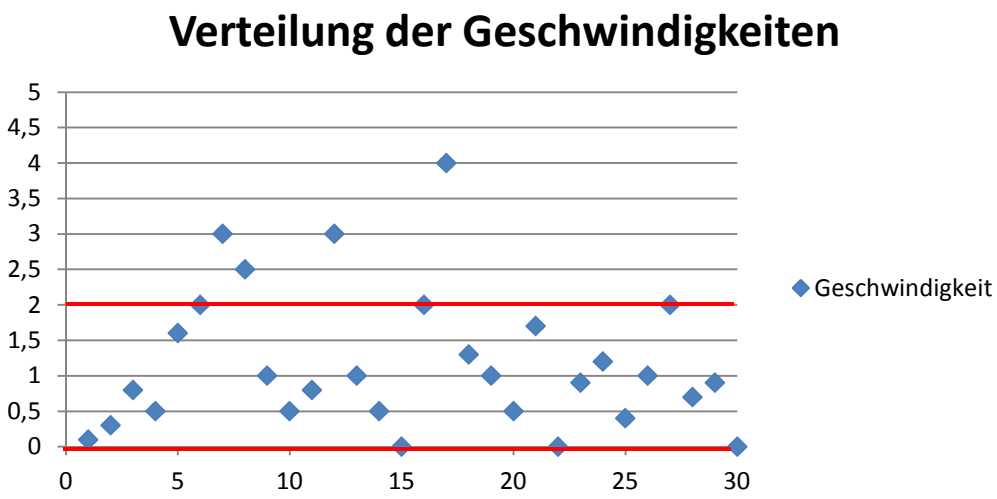


Abbildung 10: ein Beispiel für Verteilung der Geschwindigkeit

Kapitel 4

Implementierung

In diesem Kapitel wird die Implementierung dieser Ausarbeitung genau erklärt. Um den Zusammenhang zwischen Client und Server einfach verstehen zu können, wird er in der Abbildung 11 angezeigt.

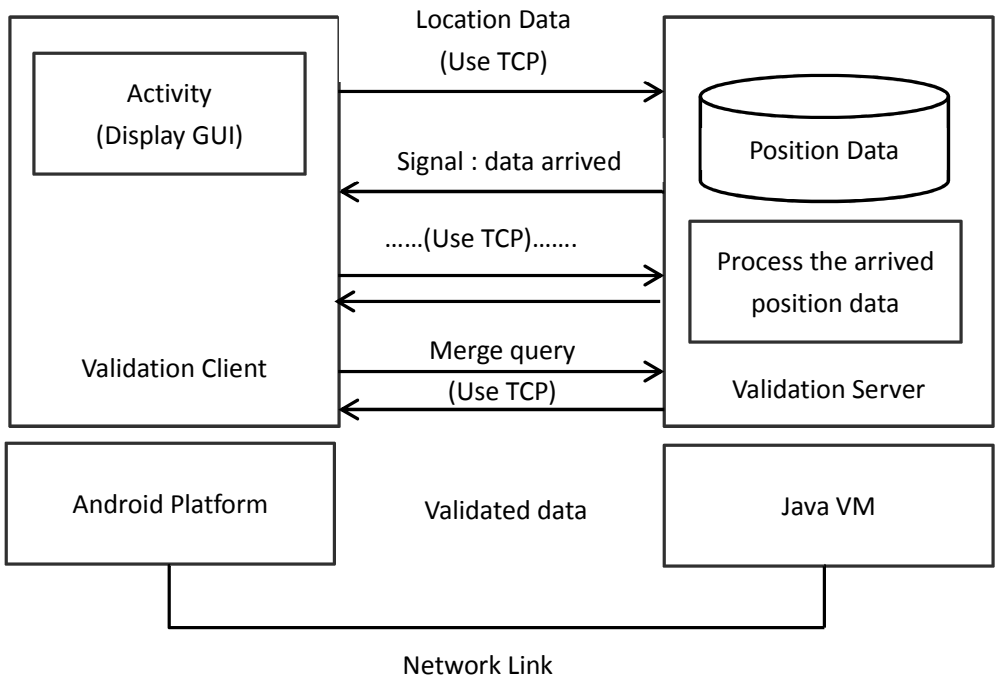


Abbildung 11: Client und Server Aufbau

Der Benutzer bringt das Mobiltelefon mit und öffnet die Applikation zum Sammeln von GPS Daten. Die erfassten GPS Traces werden durch TCP an den Validation Server gesendet. Sobald ein Trace ankommt, verarbeitet der Server die GPS Daten und speichert lokal das Ergebnis. Wenn ein Merge-Befehl von einem Server

empfangen wird, ruft dieser eine Straßenkarte auf, um die bisher nach dem Roth‘s Algorithmus erzeugten neuen Wegpunkte und validierten Wegpunkte anzuzeigen.

4.1 Validation Client auf Android System

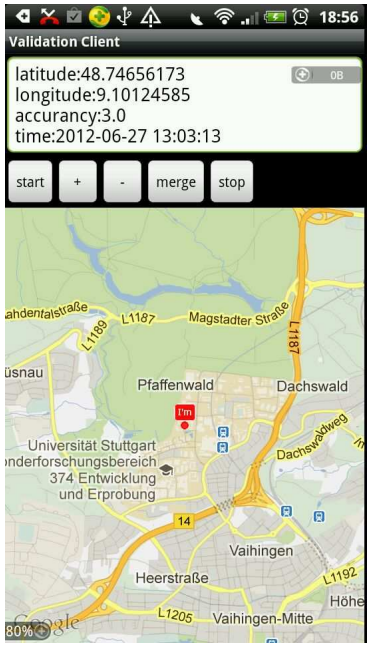


Abbildung 12: Android Benutzeroberfläche

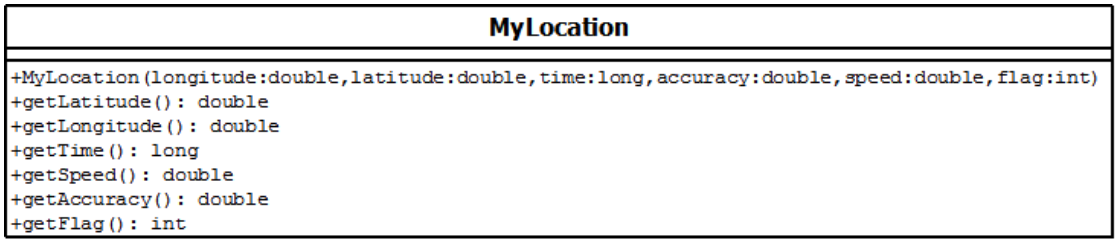
Die Hauptaufgabe der Android Applikation ist das Sammeln der GPS Traces und um diese dann zum Server zu senden. Die Benutzeroberfläche wird wie in der Abbildung 12 eingestellt.

- Ein EditText Element zeigt echtzeitig Informationen über eine aktuellen GPS Position.
- Ein MapView Element wird eingefügt, um auf Google Maps die aktuelle Position durch einen roten Punkt darzustellen. Bevor Google Maps in der Android Applikation eingebettet wird, muss man den APIKey dafür bekommen.

Hier gibt es fünf Button Elemente. Für jedes Button Element muss eine Listener-Funktion geschrieben werden, die beschreibt, was der Button nach einem Drücken tun soll.

- Der „start“-Button öffnet den GPS Sensor und fängt das Erfassen der GPS Positionen an. Die Klasse <LocationManager> bietet die Funktionen zum Zugriff auf System Location Dienst an. Durch den Dienst erhält die Applikation periodisch neue geographische Positionen des Gerätes. Die Funktion *requestLocationUpdate( )* hat einige Parameter zum Erfassen der GPS Position. *„provider“* ist der Name des Anbieters. *„minTime“* ist das minimale Zeitintervall und *„minDistance“* ist das minimale Distanzintervall zum Erfassen nächster GPS Position. Für jede GPS Position wird weiter die Geschwindigkeit zur letzten Position berechnet und lokal gespeichert. Die Geschwindigkeiten werden für eine Klassifizierung weiter genutzt.
- Der „+“-Button vergrößert die Ansicht von Google Maps.
- Der „-“-Button verkleinert die Ansicht von Google Maps.
- Der „merge“-Button sendet ein Signal zum Server, damit die Ergebnisse der Verarbeitung von den GPS Positionen auf Google Maps angezeigt und wieder am Client zurückgesendet werden.
- Der „stop“-Button schaltet den GPS Sensor aus und stoppt den „Erfassenvorgang“ der GPS Positionen.

Um die Daten sicher und sequenziell am Server zu schicken wird TCP benutzt. Das Protokoll ist ein zuverlässiges, verbindungsorientiertes, paketvermitteltes Transportprotokoll. Die Übertragungsdaten sind im Bytefluss zu transportieren. Man muss hier die Daten der GPS Positionen neu serialisieren.



Klassendiagramm 1: MyLocation Klasse implements Serializable

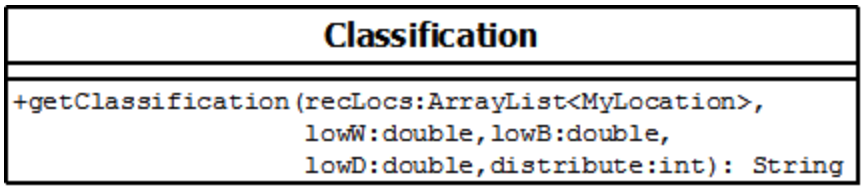
Die Klasse <MyLocation> definiert, welche Informationen eine Location enthalten soll. Die Daten über Latitude , Longitude, Accuracy und Time werden von einer GPS Location geliefert. Die Geschwindigkeit wird durch eine Division zwischen einer

Distanz zur letzten Location und das Zeitintervall erhalten. Das Flag wird als 100 eingestellt. Ein weiterer Vorteil der Klasse ist es, dass man durch *getXXX( )* Funktionen einfach die benötigten Informationen bekommen kann.

4.2 Validation Server

4.2.1 Klassifikation nach Geschwindigkeit

Nach dem Empfang eines GPS Traces wird zuerst eine Klassifikation nach der Geschwindigkeit durchgeführt. Das Ergebnis der Entscheidung ist nützlich für eine Routenbestimmung.



Klassendiagramm 2: Classification Klasse

Die Klasse <Classification> hat eine public Funktion *getClassification( )*. Sie ist für eine Klassifikation nach Geschwindigkeit verantwortlich.

- *getClassification( )* hat fünf Parameter als eingegebene Werte. Als Resultat ist eine Entscheidung in String von „walk“, „bicycle“, „drive“ oder „no decision“.
    1. *ArrayList<MyLocation> receivedLocations* ist die empfangenden GPS Daten.
    2. *double lowW* ist der niedrige Schwellenwert für Fußgänger.
    3. *double lowB* ist der niedrige Schwellenwert für Radfahrer.
    4. *double lowD* ist der niedrige Schwellenwert für Autofahrer.
    5. *double distribute* ist der niedrige Schwellenwert für die Entscheidung.

Eine Entscheidung für einen Reisemodus wird nach einer Verteilung aller Geschwindigkeiten bestimmt. Der Algorithmus wurde im Abschnitt 3.5 erklärt. Das Ergebnis wird als ein Parameter weiter genutzt, um eine Route zu bestimmen.

```
int counterW, counterB, counterD, num = numberOfCoordinates;
 for each coordinate:
      if speed in (lowW, lowB): counterW ++;
      if speed in (lowB, lowD): counter B ++;
      else : counterD ++;
      endif
 endfor
 if (counterW/num) >= distribute: return “walk”;
 if (counterB/num) >= distribute: return “bike”;
 if (counterD/num) >= distribute: return “drive”;
 else: return “no decision”;
 endif
```

Funktion *getClassification()*

4.2.2 Erfassung der Koordinaten einer Straße

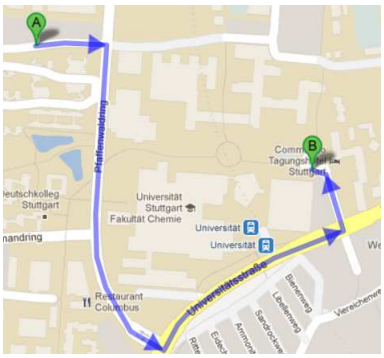
Wie kann man alle Koordinaten einer Straße bekommen?

Anfangs wurde eine Methode gefunden, indem für jede GPS Position eine Adresse auf Google Maps bestimmt wird und umgekehrt nach dieser Adresse kann man ihren geografischen Längen- und geografischen Breitengrad berechnen. Diese Methode verbraucht zu viel Zeit und Rechnen. Für eine GPS Koordinate braucht sie verdoppeltes Abfragen nach Google Maps und die Anzahl der Abfragen eines normalen Benutzers wird von Google Maps auf 2500 Male pro Tag beschränkt. Eine Übertretung kann wahrscheinlich eine Sperrung von Google Maps verursachen. Deshalb ist diese Methode sowohl zeitverschwendend als auch mühevoll. Wenn man sich langsam bewegt, sind die wiederholten Auflösungen und Umgekehrt-Auflösungen sinnlos.

Google Maps bietet ein Direction API an. Der Dienst berechnet durch eine HTTP Abfrage eine Route zwischen Standorten. Eine Abfrage ist wie *„http://maps.google apis.com/maps/api/directions/output?parameters“*. Man kann bei Bedarf mit dem Zeichen „&“ einige Parameter einfügen. *„origin“* ist z.B. eine Adresse oder

Latitude/ Longitude vom Ausgangspunkt. *„destination“* ist eine Adresse oder Latitude/ Longitude von der Endstation. *„mode“* beschreibt, was für einen Transport Modus man möchte, damit man eine angepasste Route bekommen kann. Dieser Parameter kann durch das Resultat aus Funktion *getClassification( )* ersetzt werden, so dass eine relativ exakte Route für den eingegebenen GPS Trace bestimmt werden kann. *„driving-mode“* wird als Default eingestellt. *„walking-mode“* steht nicht in allen Gebieten zur Verfügung. *„bicycling-mode“* ist zurzeit nur in USA verfügbar. Diese Routendaten werden im „XML“ oder „JSON“ Dokument ausgegeben, indem *„output“* als „XML“ oder „JSON“ eingerichtet werden. Diese Methode sieht besser aus, jedoch, nach der Beschränkung von Google Maps werden maximal acht Zwischenpunkte einer Route zurückgegeben und die Abfragen werden auch auf 2500 Male pro Tag beschränkt.

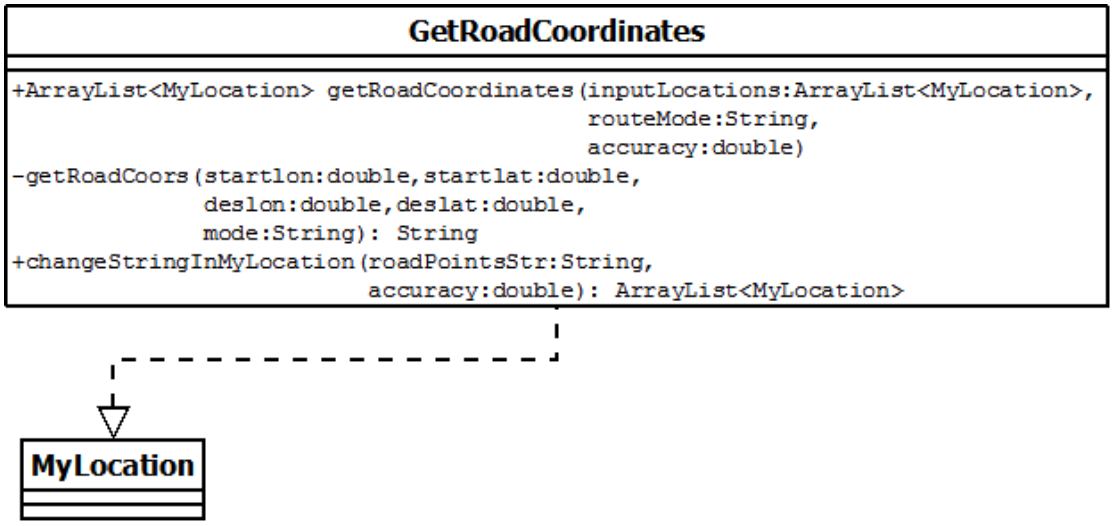
Um mehre Zwischenpunkte einer Route zu bekommen und um die Anzahl von HTTP-Abfragen nicht zu beschränken, wird eine ähnliche Methode genutzt. Der URL ist *„http://maps.google.com/maps?f=d&hl=en“*. Man kann auch durch das Einfügen der Parameter eine Route bestimmen. Aber das ausgegebene Dokument wird im KML Format geschrieben, das ähnlich wie ein XML Schema ist. Durch einen Parser vom XML extrahiert man einfach die Koordinaten einer Route.




```
- <LineString>
    <coordinates>9.101290,48.747360,0.000000 9.101920,48.747410,0.000000 9.102810,48.747380,0.000000
      9.102810,48.747380,0.000000 9.102570,48.744980,0.000000 9.102610,48.744390,0.000000
      9.102790,48.743960,0.000000 9.103340,48.743340,0.000000 9.104050,48.743070,0.000000
      9.104050,48.743070,0.000000 9.104350,48.743440,0.000000 9.104550,48.743590,0.000000
      9.105250,48.743940,0.000000 9.106400,48.744380,0.000000 9.107870,48.744760,0.000000
      9.107870,48.744760,0.000000 9.107510,48.745550,0.000000 9.107510,48.745550,0.000000
      9.107190,48.745620,0.000000 </coordinates>
  </LineString>
```

Abbildung 13: die Koordinaten einer Route von Pfaffenwaldring nach Universität Str.38. im KML Format.

Die Klasse <GetRoadCoordinates> hat folgende Aufgaben: Nach dem GPS Trace findet sie eine entsprechende Route und erzielt alle Koordinaten auf der Route.



Klassendiagramm 3: GetRoadCoordinates

- Die public Funktion *getRoadCoordinates( )* richtet die erste Koordinate der eingegebenen GPS Koordinaten als einen Startpunkt einer Route und die letzte als den Zielpunkt der Route ein. Zusätzlich wird der Routenmodus eingerichtet. Dann ruft sie die Funktion *getRoadCoors( )* auf.

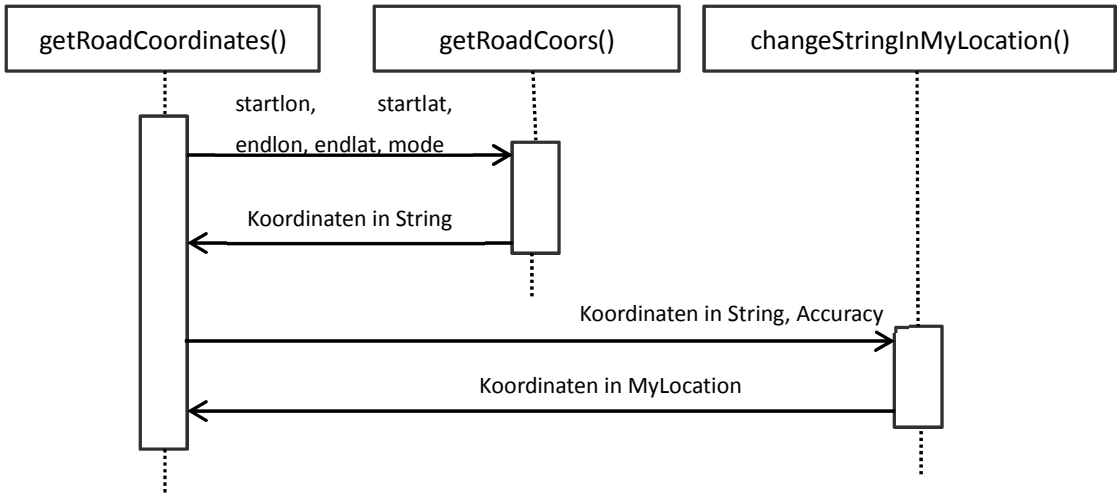


Abbildung 14: der Ablauf in der Klasse <GetRoadCoordinates>

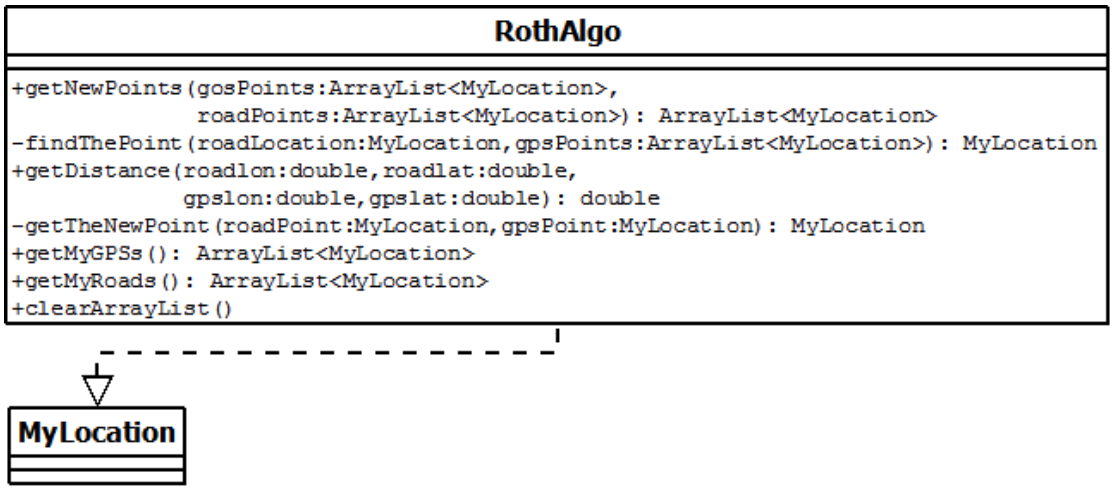
- Die private Funktion *getRoadCoors( )* bildet eine Route Abfrage mit den oben genannten Parameter nach Google Maps und erhält KML-Daten. Durch einen Parser extrahiert man alle Koordinaten zwischen den XML-Marken „LineString“ aber im String.
- Die Funktion *changeStringInMyLocation( )* trennt das Koordinatenstring auf und wandelt alle ins Double um. Mit dem Default Wert für Genauigkeit bildet man für jede Koordinate der Route den richtigen Datenaufbau.

Den ganzen Ablauf der Klasse < GetRoadCoordinates > beschreibt die Abbildung 14.

Nach einer Ausführung kann man für einen GPS Trace einige Koordinaten auf einer Straße erhalten. Danach kann die Fusionsberechnung nach dem Roth’s Algorithmus weiter durchgeführt werden.

4.2.3 Erzeugung der neuen Wegpunkte

Nun sind ein GPS Trace und die Koordinaten einer entsprechenden Straße gegeben. Nach dem Roth‘s Algorithmus im Abschnitt 3.4 kann man einen GPS Punkt und einen entsprechenden Wegpunkt verschmelzen und dadurch einen neuen Punkt erzeugen, der einen kleineren Wert über Genauigkeit besitzt. Die folgende Klasse und ihre Funktionen erledigen diese Arbeit.



Klassendiagramm 4: RothAlgo Klasse

Die Klasse <RothAlgo> hat zwei Hauptaufgaben. Erstens findet man für jede Koordinate auf Straße eine entsprechende GPS Position. Zweitens berechnet man einen neuen Punkt aus dem Paar.

- Die public Funktion *getNewPoints( )* sucht für jeden Wegpunkt nach einem Punkt auf dem GPS Trace. Wenn ein GPS Punkt dadurch gefunden wird, ruft sie die Funktion *getTheNewPoint( )* auf, um einen neuen Punkt nach dem Roth‘s Algorithmus zu erstellen.

Abbildung 15: Route und GPS Trace sind unterschiedlich

- Die private Funktion *findThePoint( )* berechnet alle Abstände zwischen einen Wegpunkt und allen Punkten auf dem GPS Trace. Der Punkt mit dem minimalen Abstand zum Wegpunkt wird ausgewählt. Weil eine Route von Google Maps automatisch berechnet wird und immer ein kürzester Pfad ausgewählt wird, ist es möglich, dass man auf einem anderen Weg gelaufen ist. In der Abbildung 15 ist der Mensch nicht entlang der grünen Linie, die Google Maps empfiehlt, sondern entlang der blauen Linie gelaufen. Auf diese Weise berechnet man einen falschen neuen Punkt. Um diese Situation bewusst zu vermeiden, stellt man eine maximale Distanz zwischen den Wegpunkten und GPS Positionen ein. Falls eine Distanz zu groß ist, vermutet man, dass das GPS Gerät wahrscheinlich nicht auf der Straße steht. Das Paar Punkte wird vernachlässigt.

- Die private Funktion *getDistance( )* berechnet eine Distanz zwischen zwei Koordinaten. Man muss darauf aufpassen, dass diese Distanz nicht auf einer Fläche, sondern auf einer Kugel liegt.

- Die private Funktion *getTheNewPoint( )* erzeugt den neuen Punkt. Die Formeln aus dem Roth's Algorithmus sind (6), (7) und (8) im Abschnitt 3.4.

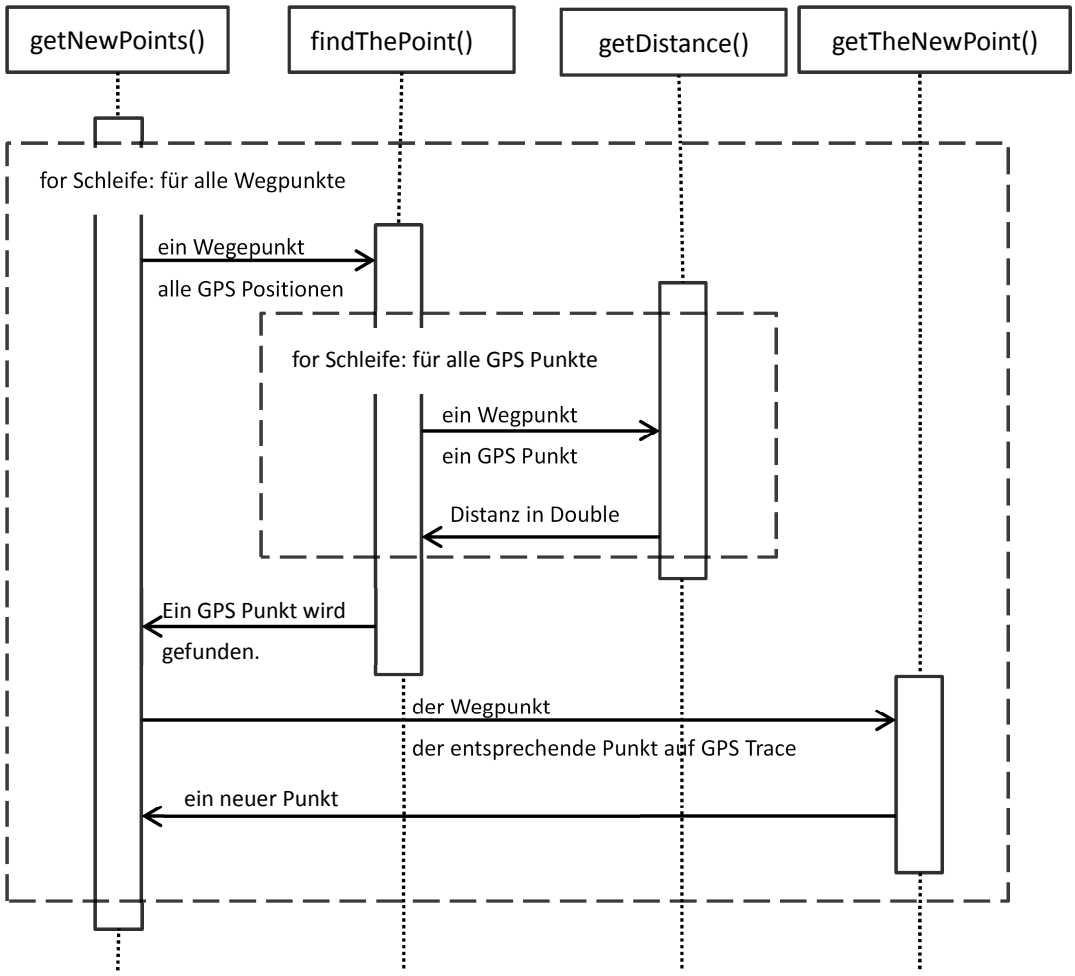


Abbildung 16: der Ablauf in der Klasse <RothAlgo>

Mit der Abbildung 16 kann man leichter den Ablauf der Funktionen verstehen.

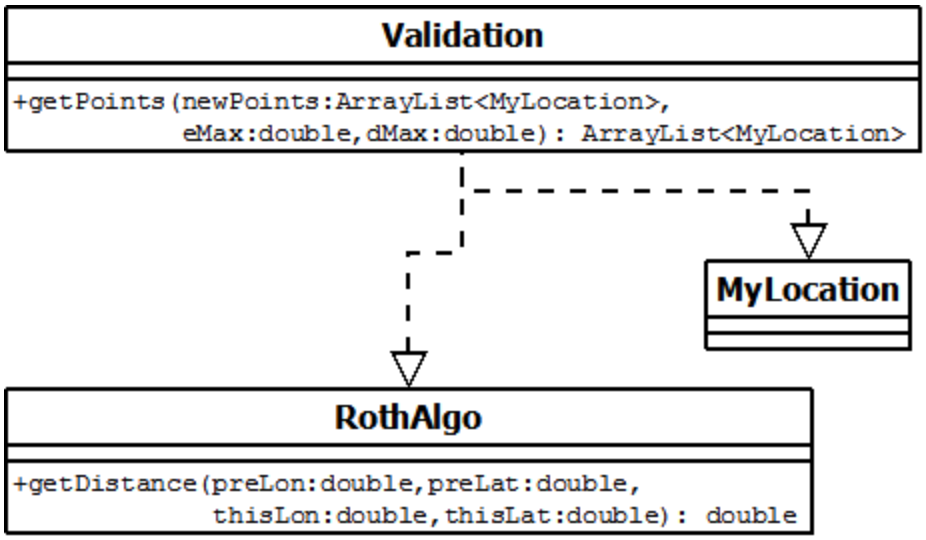
Hier werden alle neu erstellten Punkte zwischen einem erfassten GPS Trace und seiner entsprechenden Straße berechnet. Nach den Berechnungen besitzt jeder neue Punkt einen relativ kleinen Fehlerwert, der sowohl kleiner als den Fehlerwert vom entsprechenden GPS Punkt als auch kleiner als der Fehlerwert vom entsprechenden Wegpunkt ist. Falls der alte Punkt auf der Straße durch den entsprechenden neuen Punkt ersetzt wird, könnte sich die Genauigkeit einer Straßenkarte verbessern. Nachdem der Algorithmus mehrmals durchgeführt wird, wird diese Karte immer

zuverlässiger, damit sich ihre Anwendungen besser verbreiten können. Welche Segmente einer neuen Straße erfüllen die Anforderung einer Anwendung über die Genauigkeit? Man braucht eine Methode um diese Segmente zu validieren.

4.2.4 Validierung der neu erzeugten Wegpunkte

Im Abschnitt 3.2 wurde die Qualität Metriken vorgestellt. Ein validiertes Segment muss zwei Kriterien (2) und (3) erfüllen. Ein Segment ist validiert, genau dann wenn die Abstände zwischen allen aufeinander folgenden Punkten des Segments nicht größer als den maximalen Abstand sind, der vom Benutzer eingerichtet wird, und die Fehlerwerte aller Punkte auf dem Segment den gewünschten maximalen Fehlerwert nicht übertreten.

Hier entstehen alle neu erzeugten Punkte. Sie werden durch die Klasse <Validation> und ihre Funktion validiert. Die validierten Punkte existieren dann weiter.



Klassendiagramm 5: Klasse <Validation>

- In der Klasse gibt es nur eine public Funktion *getPoints ( )*. Die maximale Distanz und der maximale Fehlerwert werden hier eingestellt. Zuerst vergleicht man alle Fehlerwerte der Punkte mit dem Schwellenwert e_{max}. Die Punkte mit kleinerem Fehlerwert überbleiben. Weiter berechnet man die Distanzen zwischen einem überlebenden Punkt und seinem Nachfolger, indem die

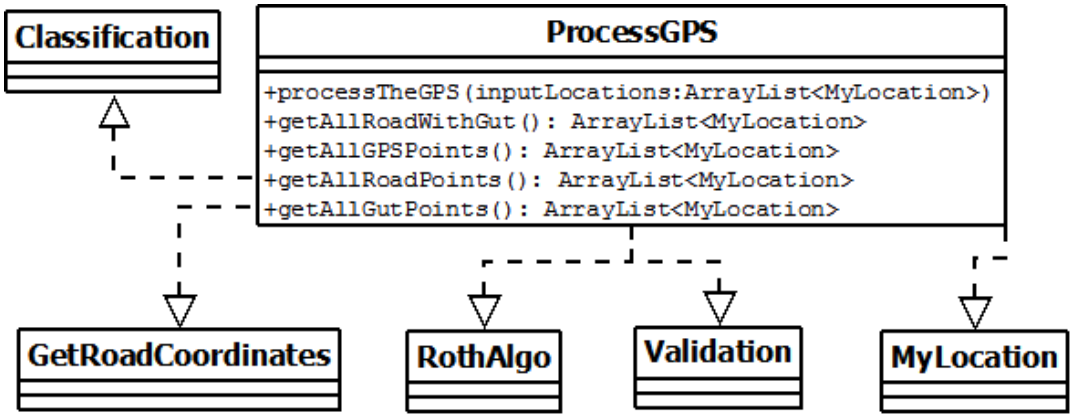
Funktion *getDistance( )* von Klasse <RothAlgo> aufgerufen wird. Wenn eine Distanz nicht den maximalen Schwellenwert d_{max} überholt, wird das Segment als „gut“ validiert. Schließlich fügt man ein Flag = 500 in den validierten Punkt ein. Das bedeutet, dass das Segment zwischen dem Punkt und seinem Vorläufer auch validiert ist.

```
for all new points pi
      if   (accuracy of pi <= eMax): save pi in ArrayList1;
 endfor
 for all points in ArrayList1
      if (distance of (pj, pj-1) <= dMax): save pj with flag = 500 in ArrayList2
 endfor
return ArrayList2
```

Funktion *getPoints ( )*

4.2.5 Verarbeitung von GPS Positionen

Die Klasse <ProcessGPS> mit ihrer Funktion *processTheGPS( )* beschreibt ein ganzes Verfahren zur Verarbeitung von empfangenen GPS Positionen. Die Funktionen in den oben geschriebenen Abschnitten werden hier in Reihenfolge aufgerufen.



Klassendiagramm 6: Klasse <ProcessGPS>

Nachdem die gesammelten GPS Positionen an den Server angekommen sind,

werden sie vor allem durch die Klasse <Classification> bearbeitet, um einen Reisemodus des Gerätes zu entscheiden. Die zurückkommende Entscheidung und der GPS Trace werden dann in die Klasse <GetRoadCoordinates> eingegeben, damit man eine entsprechende Straße finden und Informationen über die Straße bekommen kann. Nach Erhalten der Koordinaten aus der Straße, werden diese und die Koordinaten vom GPS Trace zusammen durch die Klasse <RothAlgo> berechnet, um nach dem Roth‘s Algorithmus die neuen Punkte dazwischen zu erhalten. Zuletzt validiert die Klasse <Validation> nach der Anforderung des Benutzers diese neu erstellten Punkte.

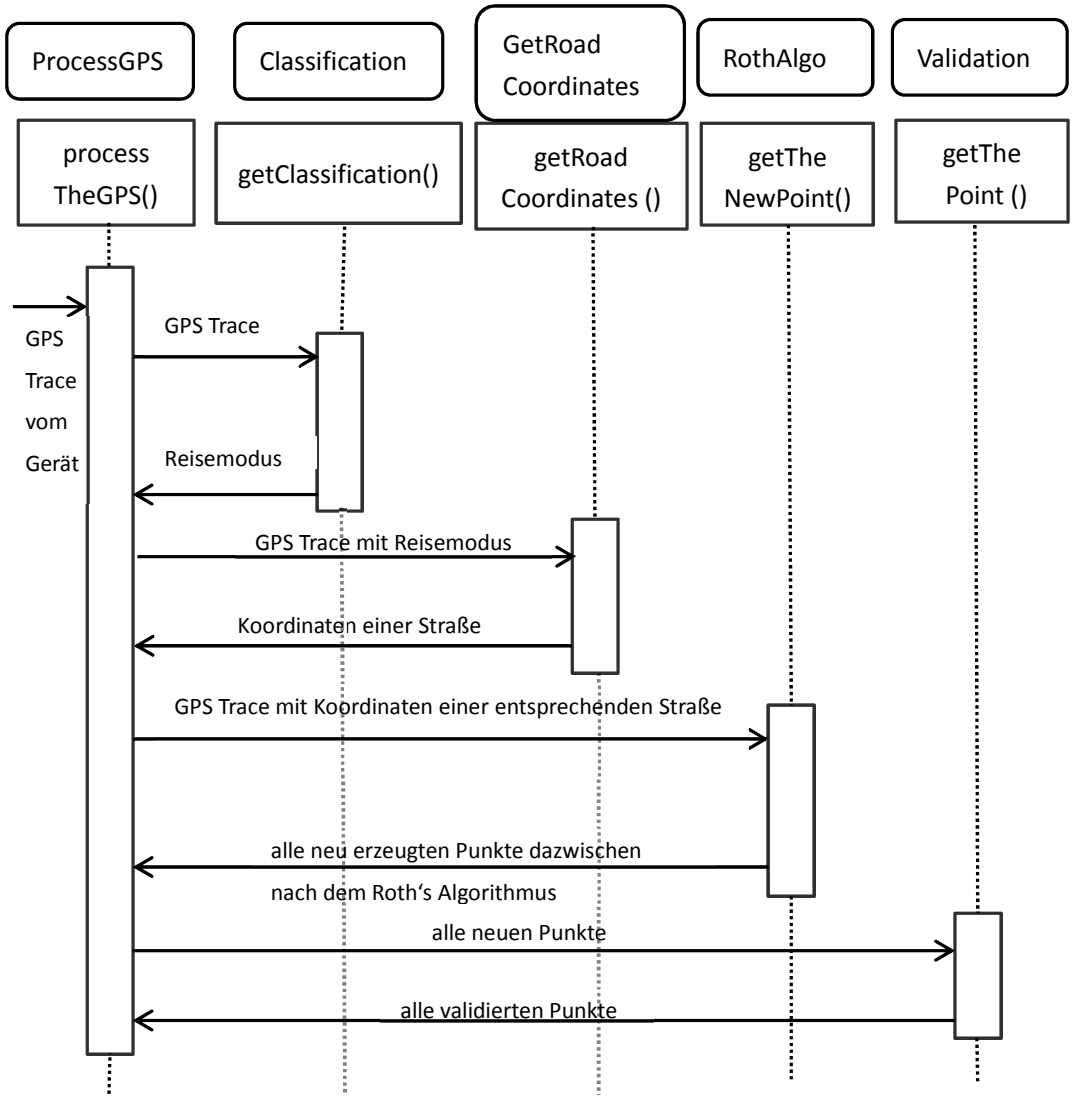


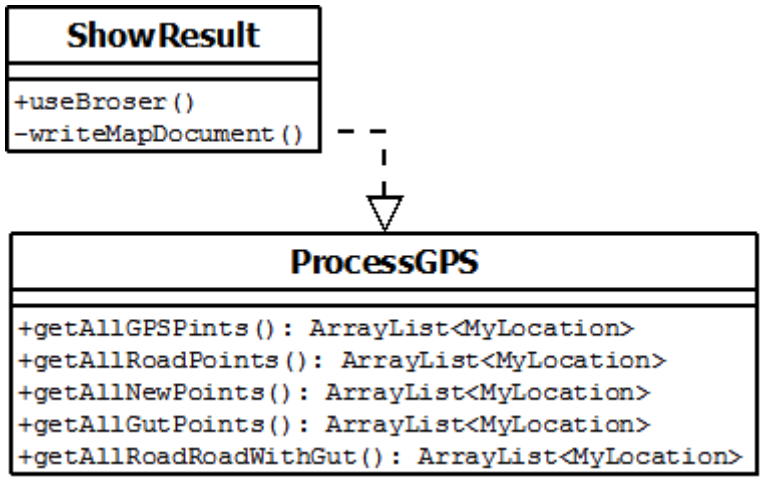
Abbildung 17: der Ablauf zwischen den Klassen

Die Abbildung 17 beschreibt den Zusammenhang zwischen den Klassen.

Alle Informationen über GPS Trace, Punkte auf Straße, neu erstellte Punkte aus dem Roth‘s Algorithmus und validierte Punkte werden lokal in ArrayList gespeichert. Sie können durch die Klasse <ShowResult> mit den Funktionen *getXXX( )* weiter benutzt werden.

4.2.6 Display der Ergebnisse

Die Klasse <ShowResult> wurde geschrieben, um nach einer Ausführung auf Google Maps die Ergebnisse zu zeigen. Damit kann man anschaulich verstehen, womit sich diese Ausarbeitung eigentlich beschäftigt. Die Klasse enthält zwei Funktionen *useBroser( )* und *writeMapDocument( ).*



Klassendiagramm 7: Klasse <ShowResult>

Durch die *getXXX( )* Funktionen der Klasse <ProcessGPS> erhält man alle benötigten Daten.

- Die private Funktion *writeMapDocument( )* schreibt alle Daten im HTML Dokument. Der GPS Trace wird durch eine blaue Linie dargestellt. Alle Koordinaten einer Straße werden durch eine grüne Linie gezeichnet. Man kennzeichnet alle neu berechneten Punkte als rote Punkte. Und falls die roten Punkte durch eine schwarze Linie verbunden sind, sind sie beide validierte Punkte.

- Die public Funktion *useBrowser( )* ruft einen Browser auf, um die von *writeMapDocument( )* erstellte Karte als eine Webseite anzuzeigen.

Bis hierher werden alle wichtigen Klassen zur Implementierung der Methoden vorgestellt. Ob sie richtig funktionieren und das Ziel der Ausarbeitung erreichen können, muss man weiter überprüfen. Es wird im nächsten Kapitel dargestellt.

Kapitel 5

Evaluation

Im Kapitel 5 wird eine Evaluation für die Methode zur Validation und das Verfahren zur Klassifikation durchgeführt. Erstens wird das Ziel der Evaluation vorgestellt. Zweitens werden die Trace Proben kurz angegeben. Dann ist die Evaluation durchzuführen. Als letztes werden die Schwächen der Implementierung diskutiert.

5.1 Ziel der Evaluation

Das erste Ziel ist für die Validationsmethode. Man muss durch ein paar Proben überprüfen, ob die von der Ausarbeitung vorgeschlagene Validationsmethode richtig durchgeführt wurde.
Das zweite ist, ob eine Vorhersage des Reisemodus nach Geschwindigkeit zur Steigerung der Effizienz der Validierung beigetragen hat.

5.2 Trace Proben für Evaluation

Zur Evaluation braucht man zuerst einige GPS Traces als Proben. Auf OpenStreetMap kann man solche Traces einfach finden, die von Teilnehmern freiwillig hochgeladen und dort als GPX Files gespeichert wurden. Ein GPX File ist ähnlich wie ein XML Dokument. Um nützliche Informationen aus dem File zu erhalten, muss man einen DOM-Parser schreiben.

Die meisten GPX Files enthalten nur drei Elemente bzw. Latitude, Longitude und die Zeit fürs Datenerfassen. Schließlich werden drei GPS Traces gefunden, die noch Elemente über Geschwindigkeit und Genauigkeit zusätzlich einschließen müssen,

weil die beiden Elemente für Implementierung notwendig sind.

	Trace 1	Trace 2	Trace 3
Anzahl der Koordinaten	25	146	285

Tabelle 1: GPS Trace Proben

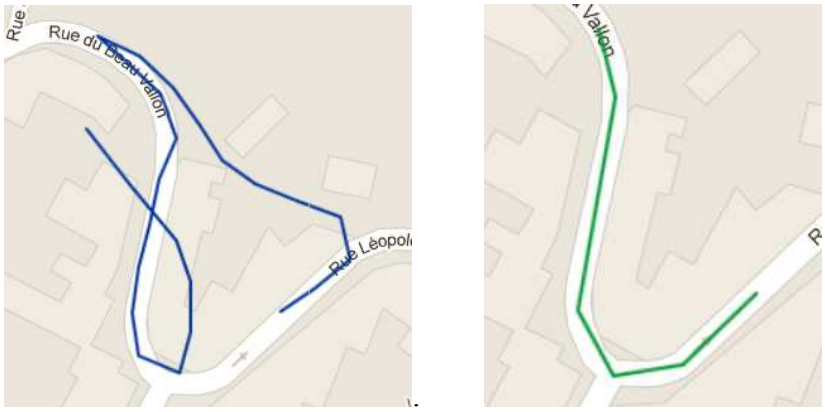
5.3 Evaluation für Validation

Um eine entsprechende Straße für jeden Trace zu finden, werden die Route Abfragen auf Google Maps durchgeführt. Durch den KML Parser erhält man für jeden Trace die Koordinaten auf einer Route.

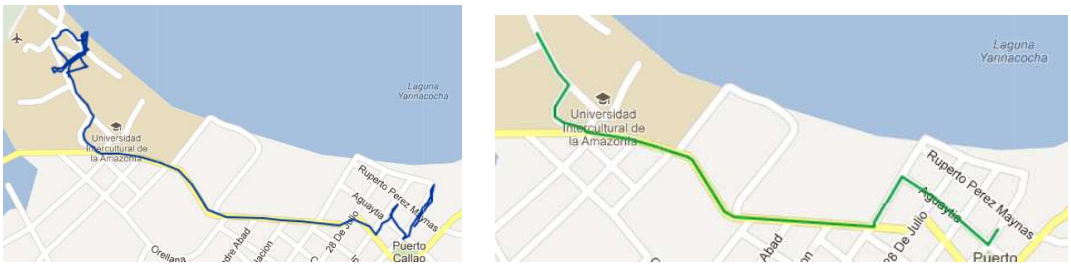
	Trace 1	Trace 2	Trace 3
Anzahl der Koordinaten der Route	7	24	32

Tabelle 2: Anzahl der Koordinaten auf eine Route

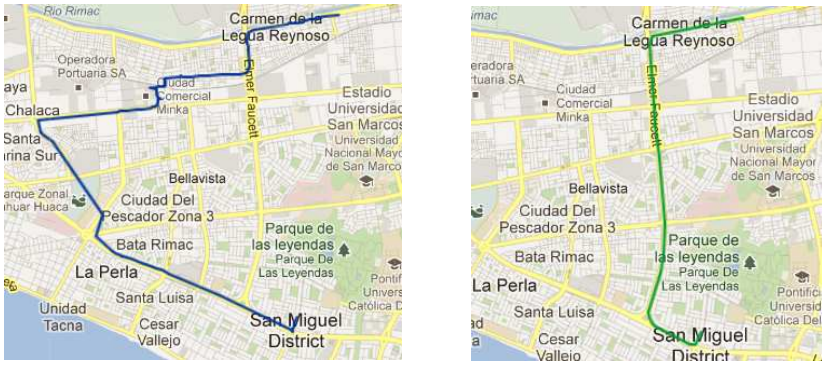
In den Tabellen 1 und 2 kann man darauf achten, dass die erzielten Koordinaten einer Route viel weniger als die auf dem GPS Trace sind. Ein wesentlicher Grund ist, dass eine Route von Google Maps automatisch ausgewählt wird und sie immer ein kürzester Pfad ist. Deshalb passt sich eine Route wahrscheinlich dem GPS Trace nicht gut an. Der GPS Trace 1 oder 3 (in der Abbildung 18a und 18c) hat nur wenige Überlappungen mit seiner Route. Eine andere Ursache ist, dass eine Route eigentlich nur wenige Informationen besitzt. Obwohl der Trace 3 (in der Abbildung 18b) seiner Route relativ gut angepasst wurde, werden hier nur 24 Koordinaten zurückgegeben.



a: Trace 1 im Blau und Route 1 im Grün



b: Trace 2 im Blau und Route 2 im Grün



c: Trace 3 im Blau und Route 3 im Grün

Abbildung 18: GPS Trace und seine entsprechende Route

Mit den Punkten auf GPS Trace und Route berechnet man nach Roth‘s Algorithmus alle neuen Punkte. Hier wird ein Schwellenwert von 30 Metern für eine maximale Distanz vom Wegpunkt zum GPS Punkt eingerichtet. Wenn der Wert übertreten wird, sind die beiden Punkte nicht angepasst und werden vernachlässigt. Hier stellt man alle Fehlerwerte der Wegpunkte als 1 Meter ein. In der Tabelle 3 steht die Anzahl der neu erstellten Punkte für jeden GPS Trace.

	Trace 1	Trace 2	Trace 3
Anzahl der Koordinaten der Route	6	16	9

Tabelle 3: Anzahl der neu erstellten Punkte nach Roth’s Algorithmus

Diese neuen Punkte werden durch die Validierungsmethode überprüft, ob sie eine Anforderung eines Benutzers erfüllen. Der maximale Abstand der Anforderung wird als 35 Metern und der maximale Wert über Genauigkeit der Anforderung als 0.99 Meter eingestellt. Die Informationen über diese validierten Segmente werden in den Tabellen 4 und 5 dargestellt.

dMax = 35 Metern, eMax = 0.99 Meter	Trace 1	Trace 2	Trace 3
Anzahl der validierten Segmente	4	5	1

Tabelle 4: Anzahl der validierten Segmente

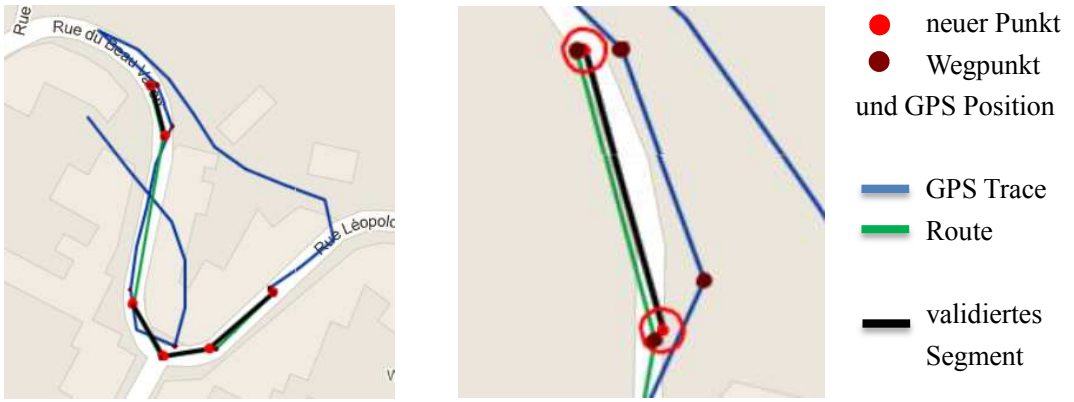
	Longitude / Latitude	Fehlerwert in Meter	Distanz zum vorgegangenen validierten Punkt in Metern
Trace 1	4.8444954246168725/ 50.47491413380044,	0.8333333333333334	13.38676823266811
	4.84448839067621/ 50.47440423624585	0.8333333333333334	15.666213136694838
	4.844655057342877/ 50.47442090291252	0.8333333333333334	11.939637149129391
	4.84488877730411/ 50.47455088757367	0.9090909090909091	21.965919539427485
Trace 2	-74.58938125166192/ 0.9267351540266893	0.944714749309973	30.66966930349886
	-74.58913909285714/ 0.9470578570997967	0.9523809523809523	33.9799970334
	-74.58843398292397/ 0.9306345915864304	0.8994395974039363	20.349980484503302
	-74.58517124898827/ 0.9428323879871339	0.9306345915864304	13.841185439980313
	-74.57951830474184/ 0.944714749309973	0.8945734975072486	13.080780756247211
Trace 3	-77.08737891975528/ -12.03790890757666	0.9351584289107764	11.117767146003814

Tabelle 5: genaue Informationen der validierten Segmente

Aus der Tabelle 5 kann man z.B. auf dem Trace 1 vier validierte Segmente finden. Jedes Segment erfüllt die Validationskriterien. Um diese klar zu erläutern werden die Segmente auf der Straßenkarte markiert (in Abbildung 19a). Die blaue Linie ist ein GPS Trace. Die Grüne ist der entsprechende Path. Ein brauner Punkt auf dem grünen Path findet einen entsprechenden braunen Punkt auf dem blauen GPS Trace. Ein roter Punkt dazwischen wird nach Roth's Algorithmus erzeugt. Falls zwei

angrenzende Punkte die Validationskriterien erfüllen, werden sie durch eine schwarze Linie verbunden. Auf der linken Seite der Abbildung 19a liegen alle validierten Segmente, die mit schwarzen Linien zusammengefügt werden.

Auf diese Weise werden die Segmente für Trace 2 und 3 analog berechnet und auf der Straßenkarte (19b und 19c) gekennzeichnet.



a: Trace 1

b: Trace 2

c: Trace 3

Abbildung 19: neu erzeugte Punkte und validierte Segmente

Daher ist es ersichtlich, dass die Methode zur Validierung in der Ausarbeitung durch die Kodierung implementiert wird.

5.4 Evaluation für Klassifikation

In der Ausarbeitung wird eine HTTP Abfrage an Google Maps gesendet, um eine Route zu erhalten, damit man einen entsprechenden Path für den eingegebenen GPS Trace bestimmen kann. Die Qualität einer Routenbestimmung beeinflusst die Erzeugung der neuen Punkte. Mit einer gut dem GPS Trace angepassten Route kann man neue Punkte effektiv berechnen.

Aber eine Route wird von Google Maps automatisch zurückgegeben und immer eine Kürzeste ausgewählt. Eine Default Route ist meistens für den Autofahrer eingestellt. Wenn man ein Fußgänger oder ein Radfahrer ist, läuft er wahrscheinlich auf einem anderen Path. Der große Teil einer Default Route ist deshalb für weitere Berechnung nutzlos. Wie kann man dieses Problem lösen? Die Abfrage einer Route bietet einen Parameter an, der den Reisemodus beschreibt. Man kann deswegen zuvor nach Geschwindigkeit des Gerätes den Reisemodus vorhersagen und den Parameter einfügen, damit eine bessere Route für den Fußgänger oder Radfahrer bekommt kann.

a b

Abbildung 20: Auswahl einer Route (Trace im Blau und Route im Grün)

Trace 2	Fußgänger	Radfahrer	Autofahrer
Verteilung der Geschwindigkeiten	130/146 = 89%	14/146 = 10%	2/146 = 1%

Tabelle 6: Klassifikation nach Geschwindigkeit

Der GPS Trace 2 besitzt zwei Routen. Eine Route ist für Autofahrer wie Abbildung 20a zeigt und eine andere ist für Fußgänger wie Abbildung 20b zeigt. Die ganze Zahl in Tabelle 6 bedeutet die Anzahl der Geschwindigkeiten. Nach der Verteilung der Geschwindigkeiten in Tabelle 6 wählt man am besten den Path in Abbildung 20b aus. Deshalb erhält man mehr als zwei validierte Segmente. Dadurch lässt sich die Effizienz der Validierung relativ erhöhen.

5.4 Schwächen der Implementierung

Wie kann man einen Pfad für den GPS Trace bestimmen und wie kann man mehre Koordinaten aus dem Pfad erhalten? Diese Probleme werden in der Ausarbeitung nicht gut gelöst.

Eine von Google Maps bestimmte Route passt sich dem GPS Trace nicht gut an. Nur kleine Teile der Route sind nützlich. Zwar wird durch eine Vorhersage des Reisemodus die Route relativ verbessert, aber nicht alle Gebiete auf Google Maps unterstützt eine Route für Fußgänger und des Weiteren bietet Google Maps eine Route für Radfahrer nur im Gebiet der USA an.

Kapitel 6

Zusammenfassung

Die Geodaten sind die grundlegenden Ressourcen aus der Natur für die auf Location basierten Dienste. Die wichtige Anwendung wie Navigation hängt stark von der Qualität dieser Daten ab. Amtliche Geodaten sind immer teuer oder geschützt durch bestimmte Lizenzgenehmigung. Heutzutage werden die Geodaten in offenen Projekten wie z.B. OpenStreetMap manuell gesammelt und unter freier Lizenz veröffentlicht. Die Methode ist aber Zeit verschwendend. Nachdem man einen GPS Sensor im Mobilgerät eingebaut hat, wird das Sammeln der Geodaten einfacher und günstiger. Die Leute können nur mit den mobilen Geräten laufen oder fahren, ohne manuelle Operationen weiter zu bedienen. Deshalb ist es möglich, umfangreiche Geodaten automatisch zu erfassen, so dass man auch die Straßenkarte automatisch erstellen kann. Aber auf diese Weise erstellte Straßenkarten besitzen eine schwache Qualität. Man kann die Karte nicht in kritischen Situationen anwenden. Um die Zuverlässigkeit der Karte sicherzustellen muss man eine Methode entwickeln, die mit Hilfe von GPS Wegpunkten diese Karte automatisch validiert.

Im Kapitel 3 der Ausarbeitung wurde diese Methode vorgestellt. Das mobile Gerät erfasst GPS Traces und sendet sie zum zentralen Server. Der Server vergleicht einen Trace mit dem Straßengraph, um eine entsprechende Straße zu finden. Danach berechnet man nach dem Roth's Algorithmus die neuen Wegpunkte zwischen dem GPS Trace und der entsprechenden Straße. Aus der Formel (8) kann man leicht herauslesen, dass im Vergleich zu den Fehlerwerten von GPS Position und Wegpunkt der Straße der Fehlerwert vom neu erstellten Punkt offensichtlich abnimmt. Das bedeutet, dass die Genauigkeit eines Wegpunktes der Straße durch eine Fusionsberechnung erhöht wird. Aber ob die aus den neuen Punkten bestehenden Segmente eine Anforderung vom Benutzer erfüllen, muss man durch eine Validation

überprüfen. Um eine subjektive Schätzung zu vermeiden, werden zwei Kriterien (2) und (3) eingeführt. Falls ein Segment als ein validiertes Segment gesehen wird, dürfen sämtliche Wegpunkte auf dem Segment die Kriterien nicht übertreten. Dieses Verfahren wird im Kapitel 4 implementiert. Zu einer anschaulichen Darstellung werden die Ergebnisse des Verfahrens auf einer Straßenkarte gezeichnet.

Eine weitere Aufgabe der Ausarbeitung ist eine Klassifizierung nach Geschwindigkeit. Im Kapitel 3 wurde eine Methode dafür entwickelt. Dort wird ein verhältnismäßiges Modell genutzt, um alle Geschwindigkeiten aufzuzeichnen. Dann kann man nach der Verteilung aller Geschwindigkeiten entscheiden, ob ein Träger des mobilen Gerätes ein Fußgänger, ein Radfahrer oder ein Autofahrer ist.

In der Evaluation wurden die bisher entwickelten Methoden durch drei Trace Proben getestet. Die Validierung daraus funktioniert und die Klassifizierung erhöht die Effizient der Validierungsmethode.

Die Routenabfrage ist eine schwache Methode zur Bestimmung einer entsprechenden Straße für den eingegebenen GPS Trace. Sie geben heutzutage nur wenige Koordinaten einer Straße zurück und die gefundene Straße passt sich dem GPS nur zum Teil an. In der Zukunft könnten die Daten auf Google Maps viel vervollständigter sein, damit man durch die Routenabfrage mehrere Informationen bekommen kann.

Zukünftig wird eine Straßenkarte automatisch erzeugt und sie besitzt gleichzeitig eine gute Qualität, die die Anforderungen der Benutzer erfüllt. Das macht das menschliche Arbeiten und das Leben immer einfacher.

Literatur

[1] Ambient-Light Sensing Optimizes Visibility and Battery Life of Portable Display Application note 5051: http://www.maxim-ic.com/an5051

[2] D. Cuff, M. Hansen, and J. Kang, “Urban sensing: out of the woods,” Commun. ACM, vol. 51, no. 3, pp. 24–33, 2008.

[3] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, “A survey of mobile phone sensing,” Communications Magazine, IEEE, vol. 48, no. 9, pp. 140–150, September 2010.

[4] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, and J. Reich, “Mobiscopes for human spaces,” IEEE Pervasive Computing, vol. 6, no. 2, pp. 20–29, 2007.

[5] M. Haklay and P. Weber, “Openstreetmap: User-generated street maps,” IEEE Pervasive Computing, vol. 7, no. 4, pp. 12–18, 2008.

[6] L. Cao and J. Krumm, “From gps traces to a routable road map,” in Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ser. GIS ’09. New York, NY, USA: ACM, 2009, pp. 3–12.

[7] R. Bruntrup, S. Edelkamp, S. Jabbar, and B. Scholz, “Incremental map generation with gps traces,” inProceedings of the 8th International IEEE Conference on Intelligent Transportation Systems, 2005.

[8] S. Morris, A. Morris, and K. Barnard, “Digital trail libraries,” in JCDL ’04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries. New York, NY, USA: ACM, 2004, pp. 63–71.

[9] Gimblett, H.R. High Lakes Inventory, Monitoring and Simulation: Salmon-Challis National Forest. Final Report Prepared for the North Fork Ranger District, Salmon-Challis National Forest. April, 2004.

[10] D. Chen, L. Guibas, J. Hershberger, and J. Sun, "Road network reconstruction for organizing paths," in Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms, 2010.

[11] J. J. Davies, A. R. Beresford, and A. Hopper, "Scalable, distributed, real-time map generation," IEEE Pervasive Computing, vol. 5, pp. 47– 54, 2006.

[12] A. Fathi and J. Krumm, "Detecting road intersections from gps traces," in Geographic Information Science, ser. Lecture Notes in Computer Science, S. Fabrikant, T. Reichenbacher, M. van Kreveld, and C. Schlieder, Eds. Springer Berlin / Heidelberg, 2010, vol. 6292, pp. 56–69.

[13] S. Minamimoto, S. Fujii, H. Yamaguchi, and T. Higashino, "Local map generation using position and communication history of mobile nodes," in Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on, April 2010, pp. 2–10.

[14] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The pothole patrol: using a mobile sensor network for road surface monitoring," in Proceeding of the 6th international conference on Mobile systems, applications, and services, ser. MobiSys '08. New York, NY, USA: ACM, 2008, pp. 29–39.

[15] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in SenSys'08: Proceedings of the 6th ACM conference on Embedded network sensor systems. New York, NY, USA: ACM, 2008, pp. 323–336.

[16] S. Rogers, P. Langley, and C. Wilson, "Mining gps data to augment road models," in KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. New York, NY, USA: ACM, 1999, pp. 104–113.

[17] P. Baier, H. Weinschrott, F. Dürr, K. Rothermel, „MapCorrect: Automatic Correction and Validation of Road Maps Using Public Sensing,“ in 2011 IEEE 36th Conference on Local Computer Networks. pp. 58-66.

[18] T. Farrell, K. Rothermel, and R. Cheng, “Processing continuous range queries with spatiotemporal tolerance,”Mobile Computing, IEEE Trans-actions on, vol. 10, no. 3, pp. 320–334, March 2011.

[19] “Navstar gps: Global positioning system standard positioning service performance standard,” United States Department of Defense, Tech. Rep., 2008.

[20] R. Lange, H. Weinschrott, L. Geiger, A. Blessing, F. Dürr, K. Rothermel, and H. Schütze, “On a generic uncertainty model for position information,” in Proceedings of the 1st International Workshop on Quality of Context. Springer, Juni 2009, Workshop-Beitrag, pp. 1–12.

[21] J. Roth, “Extracting line string features from gps logs,” Schriftenreihe der Georg-Simon-Ohm-Hochschule Nuernberg, 2008.

[22] http://de.wikipedia.org/wiki/OpenStreetMap

[23] http://openstreetmap.de/index.html

[24] http://de.wikipedia.org/wiki/Google_Maps

[25] http://de.wikipedia.org/wiki/Android_(Betriebssystem)

[26] Ed Eurnette, „Hello, Android. Introducting Google’s Mobile Development Platform“, third Edition, 2007.

[27] S. Reddy, M. Mun, J. Burke, D.Estrin, M. Hansen und M. Srivastava, „Using Mobile Phones to Determine Transportation Modes”

[28] Y. Zheng, Q. Li, Y. Chen, X. Xie und W. Ma, „Understanding Mobility Based on GPS data“

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____

Penghao Tian