Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences
Fakultät Design, Medien und Information
Department Information

Masterarbeit

RolliApp – eine GPS-basierte Android-App

Konzeption und Umsetzung eines Prototyps für die Reichweitenermittlung von elektrischen Rollstühlen

Vorgelegt von Oliver Stapelfeldt im Studiengang Informationswissenschaft und -management

Erster Prüfer: Prof. Dr. Franziskus Geeb
Zweiter Prüfer: Prof. Dr. Martin Gennis

Hammoor, Februar 2012

Abstract

Fahrer von elektrischen Rollstühlen können oftmals nicht ermitteln, welche Reichweite sie noch zurücklegen können. Der Grund ist das Fehlen einer Kilometeranzeige. Um das Problem zu lösen, wurde ein GPS-basierter Prototyp in Form einer App für das Betriebssystem Android entwickelt.

Die Gliederung dieser Masterarbeit besteht hauptsächlich aus drei Teilen. Anfänglich wird das Konzept der App beschrieben. So werden zum Beispiel die weiteren Funktionen sowie der grafische Aufbau näher erläutert. Im darauf folgenden Teil wird die Umsetzung des Konzepts behandelt. Im dritten Teil folgt schließlich die Evaluation der Umsetzung. Die Zielsetzung der Arbeit war die Entwicklung eines Prototyps, um eine Reichweitenermittlung von elektrischen Rollstühlen zu erreichen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Der Autor dieser Masterarbeit hatte sich zum Ziel gesetzt, eine Lösung für das Problem zu entwickeln, dass bei elektrischen Rollstühlen aufgrund der fehlenden Kilometeranzeige oftmals keine Aussage darüber getroffen werden kann, welche Reichweite mit der jeweiligen Akkuladung noch zurückgelegt werden kann. Bei elektrischen Rollstühlen ist im Akku die für die selbstständige Fortbewegung notwendige Energie gespeichert.

Als Lösungsansatz wurde die Entwicklung eines Prototyps für das kostenlose Smartphone-Betriebssystem Android gewählt. Da die Berechnung unter anderem der zurückgelegten Strecke mit Hilfe des GPS erfolgt, wurde zunächst das Smartphone als grundlegende Plattform ausgewählt, da in den meisten Fällen bereits ein GPS-Empfänger integriert ist. Als weitere Rahmenbedingung wurde der Prototyp für Android entwickelt, da das Betriebssystem einen hohen Marktanteil besitzt und entsprechende Smartphones in einer großen Preisspanne erhältlich sind. Diese Gründe sprechen insgesamt dafür, dass der entwickelte Prototyp (im Folgenden RolliApp genannt) für eine möglichst große Nutzerzahl zugänglich ist.

In den Kapiteln 2.1, 2.2, 2.3 und 2.4 werden zunächst die Idee hinter RolliApp und der Funktionsumfang noch einmal genauer erläutert, eine ähnliche App im Android Market untersucht sowie die Rahmenbedingungen beschrieben.

Daraufhin werden in den Kapiteln 2.5 und 2.6 sämtliche Arbeitsschritte, die für das Aufsetzen der Entwicklungsumgebung erforderlich sind, aufgeführt und die beiden Möglichkeiten zum Testen des Prototyps vorgestellt.

In den Kapiteln 2.7, 2.8, 2.9 und 2.10 folgen zunächst grundlegende Aspekte bezüglich des grafischen und funktionalen Aufbaus von RolliApp, bevor in den Kapiteln 3.1, 3.2 und 3.3 sowie 3.5, 3.6, 3.7, 3.8 und 3.9 mit der Erläuterung der jeweiligen Umsetzung fortgefahren und im Kapitel 3.11 beschrieben wird, wie eine in der Entwicklung abgeschlossene Android-App veröffentlicht wird.

Bevor im sechsten Kapitel ein Ausblick diese Arbeit damit abschließt, wie die weitere Entwicklung von RolliApp aussehen könnte, wird im fünften Kapitel noch ein Fazit gezogen, sowie im vierten Kapitel mit Hilfe einer Versuchsreihe belegt, dass die Berechnung der streckenbezogenen Funktionen korrekt funktioniert.

2 Konzept

2.1 Grundsätzliche Idee

Die Idee zu RolliApp entstand dadurch, dass beim elektrischen Rollstuhl des Autors dieser Masterarbeit keine genaue Aussage über den Ladezustand des Akkus und damit über die verbleibende Reichweite getroffen werden kann. Die Anzeige des Ladezustands erfolgt wie in Abbildung 1 dargestellt lediglich mit Hilfe von zehn farbigen Balken, deren verbleibende Anzahl den aktuellen Ladezustand widerspiegeln.

Abbildung 1: Bedienelement Rollstuhl

Der Lösungsansatz wurde in dieser Arbeit über Eingabe einer individuellen Kilometerzahl realisiert. Um erstmalig zu bestimmen, welche Reichweite mit der vollen Akkuladung zurückgelegt werden kann, sind beispielsweise mehrere vollständige Entladevorgänge beziehungsweise Fahrten mit dem Rollstuhl mit anschließender tabellarischer Auswertung denkbar. Die selbst definierte Reichweite wird daraufhin an die zurückgelegte Strecke angepasst. Wurde der Akku beispielsweise wieder aufgeladen, kann der Nutzer zusätzlich den Wert auf die vorher definierte maximale Reichweite zurücksetzen.

Ursprünglich bestand die Idee darin, dass der Nutzer optional die Möglichkeit hat, das Rollstuhlmodell auszuwählen und somit die für das jeweilige Modell entsprechende Reichweite übernommen wird. Aufgrund der gewählten Umsetzung ist RolliApp jedoch auch theoretisch für Nicht-Rollstuhlfahrer geeignet.

2.2 Funktionsumfang

Der Übersicht halber folgt zunächst die Auflistung sämtlicher Anzeigen von RolliApp, unterteilt nach Haupt- und Zusatzfunktionen:

Hauptfunktion – Verbleibende Strecke

Zusatzfunktionen – Aktuelle Geschwindigkeit, Zurückgelegte Strecke, Uhrzeit, Zeit seit Start

Zusätzlich zur Hauptfunktion bietet RolliApp weitere Funktionen an, die sowohl in der App als auch im Folgenden thematisch unterteilt sind.

Geschwindigkeit – Mit Hilfe einer ganzzahligen Kilometerangabe wird die aktuelle Geschwindigkeit des Nutzers ausgegeben.

Strecke – Bezüglich der Strecke gibt es zwei verschiedene Anzeigen. Einerseits wird die verbleibende Reichweite und andererseits die zurückgelegte Strecke ermittelt und dargestellt. Genau wie die verbleibende Reichweite lässt sich auch die zurückgelegte Strecke zurücksetzen. Der jeweils aktuelle Wert bleibt sowohl beim Pausieren als auch beim Schließen der App erhalten.

Zeit – Hier wird sowohl die Uhrzeit als auch die Zeit dargestellt, seit der die App aktiv ist. Der Nutzer hat ebenfalls die Möglichkeit, letztere Anzeige zurückzusetzen. Weiterhin bleibt der Wert beim Pausieren und Schließen der App bestehen. Die jeweilige Darstellung der Zeit (an dieser Stelle sei auf Abbildung 6 verwiesen) wurde auch aus dem Grund umgesetzt, da es möglicherweise Nutzer beziehungsweise Rollstuhlfahrer gibt, für die die Zeit beispielsweise krankheitsbedingt eine Rolle spielt.

2.3 Android Market

Beim Android Market handelt es sich ebenfalls um eine App, die einen Bestandteil des Betriebssystems darstellt. Mit ihrer Hilfe hat der Nutzer die Möglichkeit, Apps aus einem zentralen Repository herunterzuladen.

Hierfür muss sich der Entwickler zunächst registrieren und eine Gebühr in Höhe von 25 US-Dollar entrichten. Daraufhin können Apps im Android Market kostenlos angeboten werden. Sollen Apps jedoch kostenpflichtig bereitgestellt werden, ist für die Abwicklung zusätzlich die Registrierung bei Google Checkout notwendig (Google A). Für den Endkunden fällt jedoch lediglich gegebenenfalls für den Erwerb der App eine Gebühr an (Google B). Seit Anfang des Jahres 2011 ist es zudem möglich, den Android Market unter der Adresse https://market.android.com/?hl=de auch ohne Android-Smartphone zu nutzen (androidsmartphone.de).

Im Android Market lieferte eine anfängliche Suche nach dem Begriff Rollstuhl lediglich einen Treffer. Hierbei handelte es sich jedoch um eine App, die vegane Essens- und Einkaufsmöglichkeiten in Berlin auflistet. Die App wurde wahrscheinlich als ein für die Suche relevantes Ergebnis eingestuft, da im Beschreibungstext das Wort rollstuhlgerecht im Zusammenhang mit den Filterungsmöglichkeiten innerhalb der App vorhanden ist.

Die Nutzung des Suchbegriffs wheelchair ergab 24 Treffer, die jedoch ihren Namen und Beschreibungen nach alle inhaltlich ebenfalls nicht den Anforderungen entsprechen.

Da die Hauptfunktion von RolliApp, die Anzeige der verbleibenden Reichweite, auf „GPS“ basiert, erfolgte anschließend eine dementsprechende Suchanfrage mit circa 14.000 Treffern. Nun verhält es sich leider so, dass der Android Market lediglich die rudimentären Filterungsmöglichkeiten „Preis“, „Safesearch“ und „Sortieren nach“ bietet. Von den ersten beiden Filtern wurde kein Gebrauch gemacht, um keine App auszuschließen. Die Sortierung der Trefferliste nach Relevanz wurde belassen. Da es jedoch für den Autor dieser Masterarbeit unmöglich war, die große Trefferzahl zu bewältigen, wurde die erste Seite der Trefferliste (24 Treffer) herangezogen. Hierbei wurden zunächst ebenfalls die Namen und Beschreibungen betrachtet, bevor eine App (runtastic GPS Coach) genau untersucht wurde. Die übrigen 23 Treffer deuteten weiterhin auf keinen Funktionsumfang hin, der über den von runtastic GPS Coach hinausgeht.

Abbildung 2 zeigt zunächst den Hauptbildschirm der App:

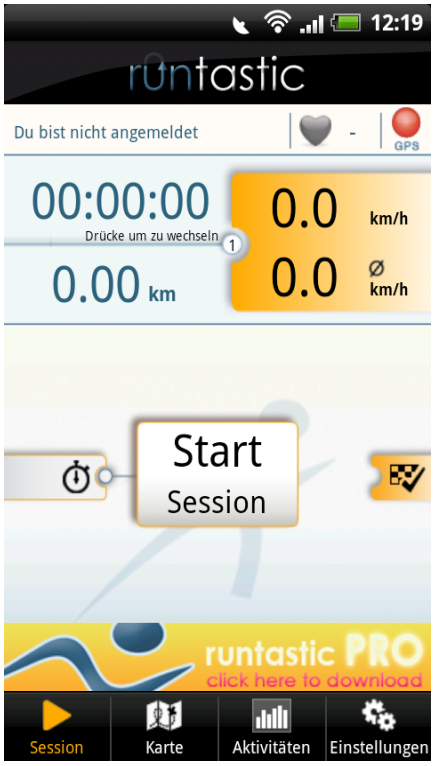

Abbildung 2: Hauptbildschirm runtastic GPS Coach

Runtastic GPS Coach ist eine Laufsport- und Fitnessapp, die vom österreichischen Unternehmen runtastic GmbH entwickelt wird. Zusätzlich wird die App auch für die Produkte IPhone, BlackBerry und Windows Phone 7 angeboten (runtastic GmbH A).

Wie in Abbildung 2 dargestellt ist, bietet runtastic GPS Coach in der kostenlosen Variante folgende Funktionen, die mit denen von RolliApp vergleichbar sind:

- Zeit seit Start
- Zurückgelegte Strecke
- Aktuelle Geschwindigkeit

Gegen eine Gebühr in Höhe von 3,99 Euro werden zusätzlich die Funktionen Sprachausgabe, Live Tracking, Verbindung mit Pulsgurt und GeoTagging unterstützt (runtastic GmbH B).

Die Anzeige der Durchschnittsgeschwindigkeit ergibt für elektrische Rollstühle keinen Sinn, da sie für gewöhnlich mit einer konstanten Höchstgeschwindigkeit betrieben werden. Aus dem Grund wurde bewusst auf die Anzeige der Durchschnittsgeschwindigkeit verzichtet.

Der Funktionsumfang von runtastic GPS Coach reicht nicht aus, um alle Anforderungen für die Untersuchung der oben genannten Aufgabenstellung durchzuführen. Für die Untersuchung wurde daher ein Prototyp entwickelt, der über die relevanten Funktionen von runtastic GPS Coach verfügt und darüber hinaus weitere relevante Funktionen, die speziell für Rollstuhlfahrer notwendig sind, ergänzt.

2.4 Rahmenbedingungen

Die Funktionen „Aktuelle Geschwindigkeit“, „Verbleibende Strecke“ und „Zurückgelegte Strecke“ benötigen jeweils den aktuellen Aufenthaltsort des Nutzers (siehe Abbildung 6). Bezüglich der Geschwindigkeit sind im Speziellen Strecke und Zeit erforderlich, da mit ihnen die Geschwindigkeit berechnet wird.

Die benötigten Informationen liefert beispielsweise das sogenannte GPS. GPS steht für Global Positioning System und wurde ursprünglich vom US-Verteidigungsministerium für militärische Zwecke entwickelt. Seit der Abschaltung der künstlichen Signalverschlechterung im Jahr 2000 hat sich GPS jedoch auch im zivilen Bereich durchgesetzt und kommt beispielsweise in Navigationssystemen zum Einsatz. Nichtsdestotrotz weist GPS im zivilen Bereich immer noch eine gewisse Ungenauigkeit auf, die bei der Umsetzung von RolliApp (Kapitel 3.6 und 3.7) eine Rolle spielte. Das System besteht aus mindestens 24 Satelliten und arbeitet mit Hilfe von Laufzeitmessungen (Köhne; Wößner). Da GPS heutzutage bereits in den meisten Smartphones integriert ist, stellen die Geräte eine geeignete Grundlage für den Prototypen dar. Die mobilen Eigenschaften des Smartphones, insbesondere das GPS, eignen sich für die Erstellung einer Softwarelösung, die als ein quasi eingebettetes System mit elektrischen Rollstühlen mitgeführt werden kann.

Als Entwicklungs- und Testumgebung wurde eine Eclipse-Umgebung mit einem Smartphone des Herstellers High Tech Computer Corporation (HTC) verwendet. Das Smartphone wurde außerdem für die Testfahrten mit dem elektrischen Rollstuhl verwendet. Das Modell hat das benötigte GPS integriert und basiert auf dem Betriebssystem Android.

Entwickelt wird Android von der Open Handset Alliance, dem Zusammenschluss von 84 Unternehmen, darunter Google, HTC, Intel und T-Mobile (Open Handset Alliance A). Android wurde für mobile Geräte konzipiert und am 21.10.2008 veröffentlicht (Google C).

Auf Android als Betriebssystem für die Entwicklungsplattform wurde aus folgenden Gründen zurückgegriffen:

Android hatte der NPD Group zufolge im zweiten Quartal 2011 einen US-Marktanteil von 52 Prozent (BIG SCREEN Internetportal). Auch bezüglich des Anschaffungspreises lässt sich eine gewisse Marktdominanz erkennen, da beispielsweise bei Amazon für die Neuanschaffung eines Android-Smartphones aufgrund der Herstellervielfalt von 65 bis 580 Euro bezahlt werden können (Stand 24.08.2011). Diese beiden Umstände tragen möglicherweise dazu bei, dass RolliApp durch die Veröffentlichung für eine große Nutzerzahl zugänglich ist. Weiterhin wurde Android als offenes Betriebssystem entworfen und baut auf einem Linux Kernel auf. Daraus ergibt sich, dass theoretisch jede Person eine App entwickeln und veröffentlichen kann, solange sie nicht gegen die Geschäfts- und Programmrichtlinien des Android Markets verstößt (Open Handset Alliance B).

2.5 Entwicklungsumgebung

Die Entwicklung einer Android-App erfolgt ausschließlich in der Programmiersprache Java. Java-Software hat den Vorteil, dass sie plattformunabhängig ist. Die Unabhängigkeit wird dadurch erreicht, dass aus dem entwickelten Code nicht der Maschinencode sondern der sogenannte Bytecode erzeugt wird. Im Gegensatz zum Maschinencode für eine spezielle Plattform (zum Beispiel Windows) steht der Bytecode, der wiederum von der sogenannten Java Virtual Machine ausgeführt wird und die zum Beispiel sowohl für Windows als auch für Mac OS verfügbar ist (Galileo Press). Demzufolge muss für ein Android-Smartphone ebenfalls die Java Virtual Machine verfügbar sein. Die Aufgabe übernimmt die Dalvik Virtual Machine. Hierbei handelt es sich jedoch streng genommen nicht um die Java Virtual Machine, da der Java-Bytecode intern noch einmal umgewandelt wird (DalvikVM.com).

Um für die Entwicklung eine vollständige Umgebung aufzusetzen sowie das in dieser Masterarbeit beschriebene Projekt zu importieren, sind folgende Arbeitsschritte notwendig:

Zunächst wird das sogenannte Java Development Kit (JDK) benötigt. Es handelt sich dabei um ein Software Development Kit (SDK), das für die Entwicklung mit Java vorausgesetzt wird und unter anderem auch eine Java-Laufzeitumgebung beinhaltet. Das JDK ist ebenfalls plattformunabhängig.

Das JDK beziehungsweise die Laufzeitumgebung ist ohnehin für die Entwicklungsumgebung Eclipse notwendig, die wiederum auf Java basiert. Mit der Hilfe von Eclipse kann Software unter anderem in Java entwickelt werden. Die Entwicklungsumgebung ist für verschiedene Plattformen und Programmiersprachen erhältlich. Für die Android-Entwicklung wird „Eclipse Classic“ empfohlen (Android.com A). Zwar wird Eclipse nicht zwingend benötigt, sondern es kann prinzipiell eine beliebige Entwicklungsumgebung genutzt werden. Eclipse bietet zusammen mit den sogenannten Android Development Tools (ADT) verschiedene Vorteile wie zum Beispiel die Entwicklung einer grafischen Nutzeroberfläche und das exportieren einer fertiggestellten App (Android.com B).

ADT ist ein Plug-in für Eclipse, das wie beschrieben verschiedene Schritte bei der Entwicklung erleichtert. Das Plug-in wird über den Update Manager von Eclipse installiert. Das erforderliche Fenster für die Installation befindet sich unter „Help>Install New Software>Add“. Durch Eingabe eines beliebigen Namens, der URL https://dl-ssl.google.com/android/eclipse/ und Bestätigen der daraufhin erscheinenden Fenster ist die Installation fast abgeschlossen. Schließlich bedarf es zusätzlich der Angabe des SDKs, die unter „Window>Preferences>Android“ zu machen ist (Android.com C).

Ob die Android-Entwicklung in Eclipse oder einer anderen Entwicklungsumgebung erfolgt, es wird auf jeden Fall das sogenannte SDK Starter Package benötigt. Dabei handelt es sich lediglich um ein Basis-SDK, das um weitere Komponenten erweitert werden muss. Zwingend erforderlich sind jedoch die SDK Platform-Tools und eine SDK Platform (Android.com D). Letztere stellt die Android-Version dar, die für die jeweilige App mindestens vorausgesetzt wird. Da RolliApp jedoch keine Funktion bietet, die eine hohe Android-Version notwendig macht, hat sich der Autor aufgrund der folgenden Statistik für die Android-Version 2.1 entschieden (Android.com E).

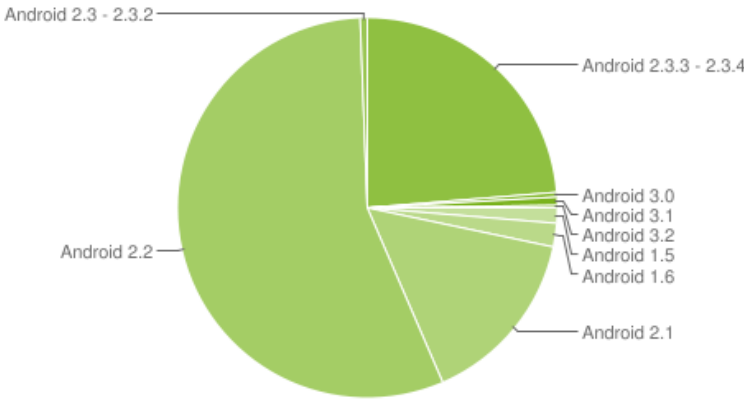

Abbildung 3: Statistik Android-Versionen

Das Importieren des Projekts dieser Masterarbeit erfolgt in Eclipse über „File>Import>General>Existing Projects into Workspace“ und anschließender Angabe des Wurzelverzeichnisses. Optional lässt sich zudem auswählen, dass eine Kopie im Workspace von Eclipse erstellt wird.

Die vollständig aufgesetzte Entwicklungsumgebung inklusive des importierten Projekts ist in Abbildung 4 dargestellt.

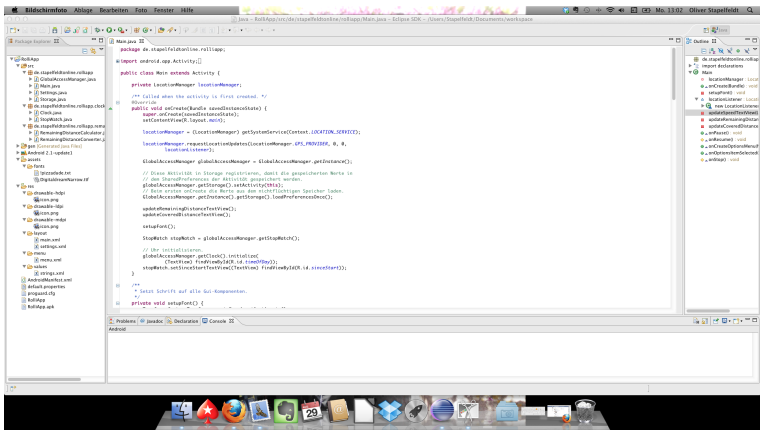

Abbildung 4: Eclipse RolliApp

Eine detaillierte Installationsanleitung ist unter der URL http://developer.android.com/sdk/installing.html zu finden.

2.6 Testmöglichkeiten

Dem Testen einer App kommt während der Entwicklung eine besondere Bedeutung zu. Grundsätzlich bestehen hierbei zwei Möglichkeiten. Einerseits enthält das SDK den Android-Emulator. Mit Hilfe des Emulators lassen sich alle Hard- und Softwarefunktionen eines Android-Smartphones außer der Telefonfunktion auch ohne ein physisches Testgerät nutzen (Android.com F). Die Einrichtung des Emulators erfolgt in Eclipse unter „Window>Android SDK and AVD Manager>Virtual devices>New“. Neben des Namens und der Größe der virtuellen SD-Karte wird die Angabe benötigt, welche Android-Version emuliert werden soll. Das SDK Starter Package muss jedoch zuvor um die Version erweitert worden sein. Bevor eine App veröffentlich wird, sollte jedoch der Test auf einem physischen Gerät erfolgen, um unter anderem sicherzustellen, dass die Nutzeroberfläche korrekt dargestellt wird (Android.com G). Abbildung 5 zeigt den Android-Emulator:

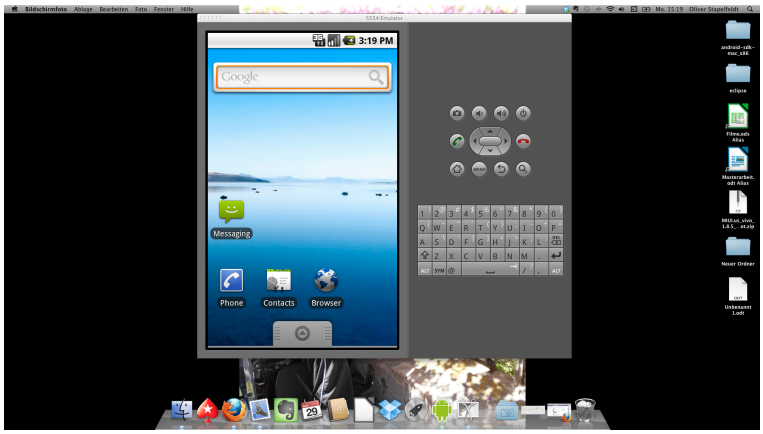

Abbildung 5: Android-Emulator

2.7 Hauptbildschirm

Abbildung 6: Hauptbildschirm

Zunächst wurde die Farbe Orange für den Hintergrund gewählt. Orange zählt zu den warmen Farben, wirkt positiv und wird weiterhin, auch bezüglich der App, mit Mode und Wandel assoziiert (Lichtkreis.at - Spirituelle Wegbegleitung durch Information, Wissen und Lichtarbeit). Als Schriftfarbe wird Schwarz verwendet, da somit zwischen den beiden Farben ein großer Kontrast besteht und eine gute Lesbarkeit gewährleistet ist (FIT für Usability. Konzeption, Design & WebBuilding by Peter Hunkirchen.). Da es sich bei RolliApp um einen erweiterten Tacho handelt, sollte die Schriftart durchgängig ein digitales Aussehen erhalten. Daher wurde die Schriftart Digital Dream Narrow von „Schriftarten.org“, die in Abbildung 9 dargestellt ist, kostenlos heruntergeladen und verwendet.

Wie bereits in Kapitel 2.2 beschrieben, werden verschiedene Funktionen geboten, die sinnvoll auf dem Hauptbildschirm angeordnet werden mussten. Da ursprünglich lediglich die Bereiche Geschwindigkeit und Strecke geplant waren, wurde der Zeitbereich am unteren Bildschirmrand positioniert. Weiterhin wurde der Geschwindigkeitsbereich aufgrund der Zahl an Anzeigen am Anfang platziert, um ein geschlossenes Gesamtbild herzustellen. Es bestand zwar zusätzlich die Möglichkeit, zwecks eines größeren Sichtbereichs die Statusleiste am oberen Bildschirmrand auszublenden, doch sie wurde unter anderem aufgrund der Empfangs- und Akkuanzeige beibehalten.

2.8 Einstellungsbildschirm

Bezüglich der Hintergrundfarbe, Schriftart und Statusleiste galten die gleichen Gründe der Umsetzung wie auch schon für den Hauptbildschirm. Lediglich in der Anordnung gibt es einige Unterschiede.

Abbildung 7: Einstellungsbildschirm

Zunächst existiert keine Geschwindigkeitseinstellung. Für die verbleibende Strecke waren sowohl ein Eingabefeld als auch eine Checkbox vorgesehen. Das Eingabefeld dient wie bereits in Kapitel 2.1 beschrieben der Eingabe einer individuellen Reichweite. Die Checkbox kann markiert werden, um den Wert zurückzusetzen. Die Möglichkeit besteht zusätzlich für die zurückgelegte Strecke sowie die Zeit seit Start. Mit Hilfe des OK-Buttons gelangt der Nutzer zurück zum Hauptbildschirm (über die Menütaste des Android-Smartphones kann der Einstellungsbildschirm aufgerufen werden).

2.9 Icon und Schriftart

Weiterhin wurde für die Darstellung der App unter anderem auf dem Home Screen eines Android-Smartphones ein Icon benötigt. Der Wunsch des Autors dieser Masterarbeit war möglichst ein modernes Rollstuhl-Motiv. Bei Fotolia, einer internationalen Bildagentur, wurde der Autor schließlich fündig und hat das Icon (siehe Abbildung 8) in einer geeigneten Ausgangs-Auflösung erworben (Fotolia). Der Hintergrund des heruntergeladenen Icons wurde schließlich mit Photoshop transparent gemacht, auf die jeweils benötigte Auflösung heruntergerechnet (siehe Kapitel 3.3) und als PNG-Datei gespeichert.

Abbildung 8: Icon

DIGITAL DREAM NARROW

Abbildung 9: Schriftart

2.10 Funktionstabelle

	1. Start	Ab 2. Start	Ende	Eingabe Verblei-bende Strecke	Reset Verblei-bende Strecke	Reset Zurückge-legte Strecke	Reset Zeit seit Start
Aktuelle Geschwin-digkeit	x	x					
Verblei-bende Strecke = 0	x						
Verblei-bende Strecke speichern			x				
Verblei-bende Strecke laden		x					
Verblei-bende Strecke = Nutzer-eingabe				x	x		
Zurückge-legte Strecke = 0	x					x	
Zurückge-legte Strecke speichern			x				
Zurückge-legte Strecke laden		x					
Uhrzeit	x	x					
Zeit seit Start = 0	x						x
Zeit seit Start speichern			x				
Zeit seit Start laden		x					

Tabelle 1: Logik

3 Umsetzung

Die im Kapitel beschriebenen Codeausschnitte stammen aus dem Eclipse-Projekt, das sich auf dem beigefügten Datenträger befindet.

3.1 Hauptbildschirm

Die grafische Umsetzung erfolgte in der HTML-ähnlichen Auszeichnungssprache Extensible Markup Language (XML) (SELFHTML e.V.). Es besteht jedoch auch die untypische Möglichkeit, die grafische Umsetzung einer Android-App direkt in Java zu tätigen (Android.com H). Somit geschah die Umsetzung des Hauptbildschirms in der Datei main.xml, die sich innerhalb der Projektstruktur im Ordner RolliApp/res/layout/ befindet (Abbildung 4).

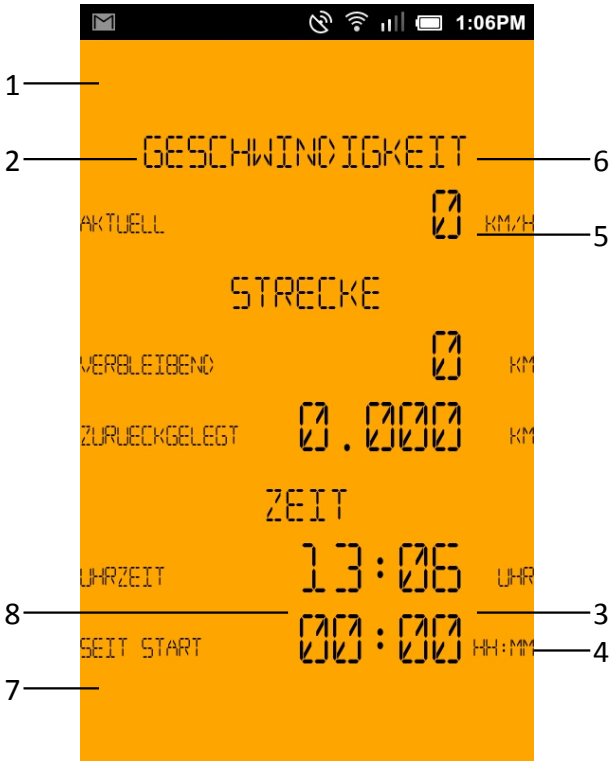

Abbildung 10: Hauptbildschirm beschriftet

1 – Hintergrundfarbe

Zunächst wurde die Hintergrundfarbe, die in Kapitel 2.7 definiert wurde, in der Entwicklungsumgebung folgendermaßen umgesetzt:

```
android:background="#FFA500"
```

Hierbei wurde der Wert für die Farbe in hexadezimaler Schreibweise angegeben. In dem Fall steht das „FF“ für einen maximalen Rotanteil, das „A5“ für einen bestimmten Grünanteil und das „00“ für keinen Blauanteil. Die Kombination ergab schließlich die gewünschte Hintergrundfarbe (dauerstress.de).

2 – Text, Textfarbe und Textgröße

Nach dem Definieren der Hintergrundfarbe erfolgte die Umsetzung der Schrift (siehe ebenfalls Kapitel 2.7), hier am Beispiel des Wortes Geschwindigkeit. Sowohl die Realisation des eigentlichen Textes als auch der Textfarbe und -größe ist im folgendem Codeausschnitt dargestellt:

```
android:text="Geschwindigkeit"
android:textColor="#000000"
android:textSize="24sp"
```

Das Besondere hierbei ist die verwendete Einheit für die Textgröße. Die Verwendung von sogenannten Scale-Independent Pixels erlaubt sowohl die Anpassung an die Dichte des jeweiligen Bildschirms als auch an die Nutzereinstellung der Schriftgröße (Android.com I).

3 – Positionierung links und unterhalb

Für die grafische Umsetzung kamen auch direkte Positionsangaben zur Anwendung, im folgendem Beispiel für den Wert Zeit seit Start:

```
android:layout_toLeftOf="@+id/sinceStartUnit"
android:layout_below="@+id/timeOfDay"
```

Das Beispiel verdeutlicht außerdem, warum es notwendig ist, dass jedem Element eine eindeutige ID zugewiesen wird.

4 – Ausrichtung ganz rechts und unten

Zusätzlich zur direkten Positionierung wurde im folgenden Beispiel auch die Einheit von „Zeit seit Start“ an jener ausgerichtet:

```
android:layout_alignParentRight="true"
android:layout_alignBaseline="@+id/sinceStart"
```

Im Beispiel wurde zunächst mit Hilfe der booleschen Variable das Element am rechten Rand des Eltern-Elements, hier der rechte Bildschirmrand, ausgerichtet. Zusätzlich wurde für die Ausrichtung an der Unterkante des Elements mit der ID sinceStart gesorgt.

5 – Ausrichtung rechts

Der Vollständigkeit halber ist im Folgenden exemplarisch die Ausrichtung des Wertes für die aktuelle Geschwindigkeit am Element mit der ID sinceStart aufgeführt:

```
android:layout_alignRight="@+id/sinceStart"
```

6 – Horizontale Zentrierung

Mit Hilfe des folgenden Codeausschnitts wurde für die horizontale Zentrierung, hier des Wortes Geschwindigkeit, gesorgt:

```
android:layout_centerHorizontal="true"
```

7 – Vertikale Zentrierung

Demgegenüber musste die vertikale Zentrierung im Folgenden lediglich einmal für das gesamte Eltern-Element definiert werden:

```
android:gravity="center_vertical"
```

8 – Abstand oben und rechts

Schließlich zeigt folgendes Beispiel, in dem Fall der Wert Zeit seit Start, die Verwendung von Abständen:

```
android:layout_marginTop="12dip"
android:layout_marginRight="6dip"
```

Bei der Einheit dip (oder auch dp) handelt es sich um sogenannte Density-Independent Pixels, die immer dann verwendet werden sollte, wenn keine Schriftgröße definiert wird.

3.2 Einstellungsbildschirm

Die Umsetzung erfolgte in der XML-Datei RolliApp/res/layout/settings.xml (Abbildung 4).

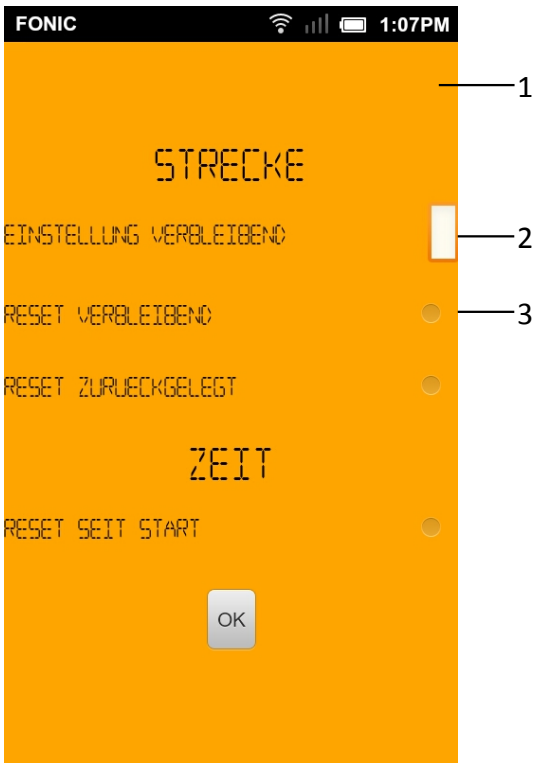

Abbildung 11:
Einstellungsbildschirm beschriftet

1 – Eingabefeld

Für den Einstellungsbildschirm wurde zunächst das Eingabefeld mit den Besonderheiten, dass es sich um ein Nummern-Eingabefeld mit einer maximalen Länge von zwei Nummern handelt, folgendermaßen umgesetzt.

android:inputType="number"

android:maxLength="2"

2 – Checkbox

Außerdem wurden mehrere Checkboxen durch Verwendung von „<CheckBox/>“ realisiert.

3 – Knopf

Schließlich wurde mit Hilfe von „<Button/>“ ein Knopf umgesetzt.

3.3 Icon und Schriftart

Bezüglich des Icons war die Grundvoraussetzung, dass das PNG-Format verwendet wird (Android.com J). Da Android-Smartphones über verschiedene Bildschirm-Dichten verfügen, sollte das gewünschte Icon jeweils in der Auflösung 72x72 px, 48x48 px sowie 36x36 px vorliegen (Android.com K). Die bearbeiteten Icons wurden daraufhin in die jeweils dafür vorgesehenen Ordner innerhalb der Projektstruktur von Eclipse kopiert. Dabei handelte es sich um die Ordner drawable-hdpi, drawable-ldpi und drawable-mdpi unterhalb von „RolliApp/res“ (Abbildung 4). Schließlich musste das Icon in der Datei AndroidManifest.xml angegeben werden. Hierzu benötigte jedes der drei Icons den gleichen Dateinamen, damit der folgende Codeausschnitt korrekt funktioniert:

```
android:icon="@drawable/icon
```

Die gewünschte Schriftart für den Haupt- und Einstellungsbildschirm lag im TTF-Format vor. Die Datei wurde daraufhin in den Ordner RolliApp/assets/fonts/ kopiert und mit Hilfe der folgenden Methode auf die entsprechenden Elemente, hier das Wort Geschwindigkeit, angewendet:

```
private void setupFont() {
Typeface font = Typeface.createFromAsset(getAssets(),
"fonts/DigitaldreamNarrow.ttf");

TextView speedAreaTextView = (TextView) findViewById(R.id.speedArea);

speedAreaTextView.setTypeface(font);
}
```

Innerhalb der Methode setupFont wurde zunächst das Typeface-Objekt „font“ erzeugt und diesem die Datei DigitaldreamNarrow.ttf zugewiesen. Zusätzlich wurde das Textview-Objekt speedAreaTextView erschaffen und diesem das Element mit der ID speedArea zugewiesen. Schließlich wurde die gewünschte Schriftart auf alle entsprechenden Elemente des Haupt- und Einstellungsbildschirms angewendet.

3.4 Zusammenfassung der grafischen Umsetzung

Verwendung	**Attribut**
Abstand nach oben	android:layout_marginTop
Abstand nach rechts	android:layout_marginRight
Ausrichtung am rechten Rand	android:layout_alignParentRight
Ausrichtung am rechten Rand von ID	android:layout_alignRight
Ausrichtung am unteren Rand von ID	android:layout_alignBaseline
Hintergrundfarbe	android:background
Horizontale Zentrierung	android:layout_centerHorizontal
Icon	android:icon
ID	android:id
Maximale Länge des Eingabefelds	android:maxLength
Positionierung links von ID	android:layout_toLeftOf
Positionierung unterhalb von ID	android:layout_below
Text	android:text
Textfarbe	android:textColor
Textgröße	android:textSize
Typ des Eingabefelds	android:inputType
Vertikale Zentrierung	android:gravity

Tabelle 2: Zusammenfassung Umsetzung Haupt- und Einstellungsbildschirm

3.5 Aktuelle Geschwindigkeit

Die Datei Main.java, die unter anderem den entsprechenden Code beinhaltet, befindet sich im Ordner RolliApp/src/de.stapelfeldtonline.rolliapp/.

```
private void updateSpeedTextView(Location location) {
TextView speedTextView = (TextView) findViewById(R.id.speed);

Float speedTextViewFloat = location.getSpeed() * 3.6f;
String speedTextViewString = String.format("%.0f", speedTextViewFloat);
speedTextView.setText(speedTextViewString);
}

public void onLocationChanged(Location location) {
updateSpeedTextView(location);
}
```

Zunächst wurde innerhalb der Methode updateSpeedTextView das Textview-Objekt speedTextView erzeugt und diesem das Element mit der ID speed zugewiesen. Anschließend erfolgte die Berechnung und Darstellung des eigentlichen Wertes der aktuellen Geschwindigkeit. Einerseits wurde hierfür das Float-Objekt speedTextViewFloat erschaffen und diesem mit Hilfe der getSpeed-Methode die aktuelle Geschwindigkeit zugewiesen. Zusätzlich folgte eine Multiplikation, da der ursprünglich zurückgegebene Wert die Einheit m/s besitzt. Andererseits wurde dieser Wert daraufhin formatiert, umgewandelt und wiederum dem String-Objekt speedTextViewString zugewiesen. Schließlich wurde der Wert auf das eingangs erzeugte Textview-Objekt angewendet.

Die Methode onLocationChanged sorgt daraufhin dafür, dass bei jeder Änderung des Aufenthaltsorts die zuvor beschriebene Methode updateSpeedTextView aufgerufen wird.

3.6 Verbleibende Strecke

Der folgende Codeausschnitt befindet sich ebenfalls in der Datei Main.java und der darauf folgende in der Datei RolliApp/src/de.stapelfeldtonline.rolliapp.remainingdistance/Remaining-DistanceCalculator.java.

Darstellung

```
private void updateRemainingDistanceTextView() {
TextView remainingDistanceTextView = (TextView)
findViewById(R.id.remainingDistance);

float remainingDistance =
RemainingDistanceConverter.convertMetersToKilometers(GlobalAccessMana-
ger.getInstance().getRemainingDistanceCalculator().getRemainingDistance());
String remainingDistanceString = String.format("%.0f", remainingDistance);
remainingDistanceTextView.setText(remainingDistanceString);
}

public void onLocationChanged(Location location) {
GlobalAccessManager.getInstance().getRemainingDistanceCalculator().update-
CurrentLocation(location);

updateRemainingDistanceTextView();
}
```

Zunächst ist die Methode updateRemainingDistanceTextView ähnlich zur im vorangegangenen Kapitel beschriebenen Methode updateSpeedTextView aufgebaut.

Ebenso verhält es sich mit der Methode onLocationChanged, mit dem Unterschied, dass die eigentliche Berechnung der verbleibenden Reichweite nicht direkt in dieser Java-Datei sondern innerhalb der externen Methode updateCurrentLocation geschieht. Zu dem Zweck wurde ein GlobalAccessManager verwendet, der den Zugriff auf verschiedene Dateien ermöglicht.

Berechnung

```
public void updateCurrentLocation(Location currentLocation) {
if (lastLocation != null) {
float distance = lastLocation.distanceTo(currentLocation);
float accuracy = currentLocation.getAccuracy();

if (distance >= accuracy) {
remainingDistance -= distance;

this.lastLocation = currentLocation;
}
} else {
this.lastLocation = currentLocation;
}
}
```

```
public float getRemainingDistance() {
return Math.max(remainingDistance, 0);
}
```

Die Methode updateCurrentLocation enthält zwei If- und eine Else-Anweisung. Immer dann wenn die sogenannte lastLocation ungleich null ist, werden zwei Befehle ausgeführt. Zum einen wird die Distanz zwischen der last- und der currentLocation vom Float-Wert distance subtrahiert. Zum anderen wird die Akkuratheit der currentLocation abgefragt. Weiterhin wird geprüft, ob die berechnete Distanz größer als die Akkuratheit ist und gegebenenfalls die Distanz von der gesamten verbleibenden Reichweite subtrahiert sowie die current- als lastLocation gespeichert. Andernfalls wird lediglich die current- als lastLocation gespeichert.

Die Akkuratheit hängt mit der Genauigkeit des GPS-Signals zusammen. Das Signal hat eine bestimmte Ungenauigkeit, die dazu führt, dass obwohl das Smartphone nicht vom Ort wegbewegt wird, der GPS-Empfänger bei wiederholten Messungen dennoch ungenaue Koordinaten in einem bestimmten Umkreis liefert. Die Akkuratheit gibt hierbei an, welche Abweichungen von der echten Position des Smartphones maximal auftreten können. Werden zwei Messwerte verglichen und der zweite liegt innerhalb des Ungenauigkeits-Umkreises, kann keine genaue Aussage darüber getroffen werden, ob das Smartphone seine Position geändert hat. Liegt die zweite Messung jedoch außerhalb des Umkreises, dann ist es wahrscheinlich bewegt worden.

Die Methode getRemainingDistance gibt schließlich den Wert remainingDistance zurück, der wiederum in der eingangs beschriebenen Methode updateRemainingDistanceTextView benötigt wird.

3.7 Zurückgelegte Strecke

Da die Funktionen bezüglich der Strecke sehr ähnlich zueinander aufgebaut sind, befindet sich auch der entsprechende Codeausschnitt in den beiden selben Dateien, die bereits zuvor genannt wurden.

Darstellung

```
private void updateCoveredDistanceTextView() {
TextView coveredDistanceTextView = (TextView)
findViewById(R.id.coveredDistance);

float coveredDistance =
RemainingDistanceConverter.convertMetersToKilometers(GlobalAccessMana-
ger.getInstance().getRemainingDistanceCalculator().getCoveredDistance());
String coveredDistanceString = String.format("%.3f", coveredDistance);
coveredDistanceTextView.setText(coveredDistanceString);
}

public void onLocationChanged(Location location) {
GlobalAccessManager.getInstance().getRemainingDistanceCalculator().update-
CurrentLocation(location);

updateCoveredDistanceTextView();
}
```

Die Methode updateCoveredDistanceTextView unterscheidet sich im Wesentlichen zur entsprechenden Methode im vorangegangenen Kapitel dadurch, dass der in der Berechnung zurückgegebene Wert coveredDistance mit Hilfe der Methode getCoveredDistance in der vorgesehenen Textview dargestellt wird.

Bei jeder Änderung des Aufenthaltsorts wird wiederum die entsprechende Update-Methode aufgerufen.

Berechnung

```
public void updateCurrentLocation(Location currentLocation) {
if (lastLocation != null) {
float distance = lastLocation.distanceTo(currentLocation);
float accuracy = currentLocation.getAccuracy();

if (distance >= accuracy) {
coveredDistance += distance;

this.lastLocation = currentLocation;
}
} else {
this.lastLocation = currentLocation;
}
}
```

```
public float getCoveredDistance() {
return coveredDistance;
}
```

Die Berechnung der zurückgelegten Strecke erfolgt dabei innerhalb der Methode updateCurrentLocation durch Addition der ermittelten Distanz mit dem Wert coveredDistance. Das geschieht ebenfalls jedoch immer nur dann, wenn die entsprechende Distanz größer als die Akkuratheit ist. Diese Überprüfung wurde aus dem Grund umgesetzt, um Ungenauigkeiten bezüglich der Positionsbestimmung weitestgehend zu vermeiden. Beispielsweise würde eine Änderung des Aufenthaltsorts um einen Meter bei einer Akkuratheit von mehr als einem Meter ignoriert werden. Die Einschränkung wurde jedoch zwecks einer größeren Messgenauigkeit in Kauf genommen.

3.8 Uhrzeit

Die beiden im Wesentlichen für die Uhrzeit zuständigen Methoden befinden sich in der Datei RolliApp/src/de.stapelfeldtonline.rolliapp.clock/Clock.java.

```
private void updateGui() {
CharSequence timeString = DateFormat.format("kk:mm",
System.currentTimeMillis());

if (timeOfDayTextView != null) {
timeOfDayTextView.setText(timeString);
}
}

public void resume() {
if (handler == null) {
handler = new Handler();
handler.postDelayed(task, REFRESH_TIME);
}

updateGui();
}
```

Zunächst sorgt die Methode updateGui dafür, dass die aktuelle Uhrzeit des Betriebssystems auf die entsprechende Textview angewendet wird. Hierfür wird im Detail das Charsequence-Objekt timeString erzeugt, diesem die formatierte Uhrzeit zugewiesen und das Objekt daraufhin auf die Textview timeOfDayTextView angewendet.

Die Methode resume ist der zuvor beschrieben Methode insofern übergeordnet, als dass sie beispielsweise das Aktualisierungsintervall REFRESH_TIME beinhaltet. Die Aktualisierung findet minütlich statt. Das bedeutet jedoch auf der anderen Seite, dass ein Sprung zur folgenden Minute innerhalb der App nicht unbedingt synchron zur tatsächlichen Uhrzeit erfolgen muss.

3.9 Zeit seit Start

Der Codeausschnitt befindet sich in der Datei RolliApp/src/de.stapelfeldtonline.rolliapp.clock/StopWatch.java.

```
private void updateGui() {
long elapsedTime = getElapsedTime();

long minutes = (elapsedTime / 60000) % 60;
long hours = (elapsedTime / 60000) / 60;

CharSequence timeString = String.format("%02d", hours) + ":" +
String.format("%02d", minutes);

if (sinceStartTextView != null) {
sinceStartTextView.setText(timeString);
}
}

public void run() {
updateGui();

handler.postDelayed(this, REFRESH_TIME);
}
```

Die Methode updateGui ist für die schlussendliche Darstellung der „Zeit seit Start“ zuständig. Hierfür wird zunächst der Long-Wert elapsedTime erzeugt und diesem der Wert der Methode getElapsedTime zugewiesen. Daraufhin wird der Wert, der ursprünglich in Millisekunden vorliegt, entsprechend umgerechnet und jeweils wiederum als Long-Werte minutes beziehungsweise hours gespeichert. Schließlich werden die Werte formatiert, dem Charsequence-Objekt timeString zugewiesen und auf die Textview sinceStartTextView angewendet.

Die Methode getElapsedTime wurde umgesetzt, um bei einer Unterbrechung der Messung zu erreichen, dass für die Berechnung die Unterbrechung nicht stattgefunden hätte.

Die Methode run, die wiederum die Methode updateGui aufruft, wird minütlich ausgeführt.

3.10 Verschiedenes

Zum einen handelt es sich um den sogenannten Locationmanager, der für die Kommunikation mit dem GPS-Empfänger verantwortlich ist. Zum anderen wird am Beispiel des Wertes für die zurückgelegte Strecke der Speicher- und Ladevorgang beschrieben. Hierbei wird der Locationmanager lediglich in der Datei RolliApp/src/de.stapelfeldtonline.rolliapp/Main.java erzeugt. Demgegenüber erfolgte die Umsetzung des Speicher- und Ladevorgangs in der Datei RolliApp/src/de.stapelfeldtonline.rolliapp/Storage.java.

```
locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
```

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0,
0, locationListener);
```

Zunächst wird das Locationmanager-Objekt locationManager erzeugt und diesem der Dienst LOCATION_SERVICE des Betriebssystems zugewiesen. Anschließend wird regelmäßig mit Hilfe der Methode requestLocationUpdates, ohne dass eine bestimmte Zeit und Distanz vergangen beziehungsweise zurückgelegt sein muss, eine Aktualisierung des Aufenthaltsorts durchgeführt.

```
public void savePreferences() {
float distanceCovered = remainingDistanceCalculator.getCoveredDistance();

savePreferences(COVERED_DISTANCE, distanceCovered);
}
```

```
public void loadPreferences() {
SharedPreferences sharedPreferences =
activity.getPreferences(Context.MODE_PRIVATE);

if (sharedPreferences.contains(COVERED_DISTANCE)) {
float coveredDistance = sharedPreferences.getFloat(COVERED_DISTANCE, 0f);
remainingDistanceCalculator.setCoveredDistance(coveredDistance);
}
}
```

Um den Float-Wert für die zurückgelegte Strecke zu speichern, wird diesem innerhalb der Methode savePreferences der entsprechende Wert aus der Methode getCoveredDistance aus dem „remainingDistanceCalculator“ zugewiesen. Anschließend wird der Wert unter dem Eintrag COVERED_DISTANCE gespeichert.

Schließlich sorgt die Methode loadPreferences dafür, dass der entsprechende Wert wieder geladen wird. Falls die sogenannten sharedPreferences den Wert COVERED_DISTANCE enthalten, wird er wieder ausgelesen, dem Float-Wert coveredDistance zugewiesen und mit Hilfe der Methode setCoveredDistance im „remainingDistanceCalculator“ weiterverwendet.

3.11 Veröffentlichung im Android Market

Der erste Schritt besteht aus dem Signieren. Das ist erforderlich, da Android aufgrund der Vertrauenswürdigkeit voraussetzt, dass jede zu installierende App signiert sein muss. Durch das Zertifikat ergeben sich jedoch weder für den Entwickler noch für den Nutzer einer App Nachteile. Das bedeutet, dass das Zertifikat durch den Entwickler ausgestellt werden kann und dass durch das Zertifikat nicht geregelt wird, ob eine App installiert werden darf oder nicht (Android.com L). Sobald die Entwicklung einer App abgeschlossen ist, bieten die ADT eine einfache Möglichkeit, um den Exportprozess, der unter anderen das Signieren umfasst, durchzuführen. Zunächst muss in Eclipse „File>Export“ ausgewählt werden. Der Export, für den die Auswahl des Projekts und die Erzeugung des Keystores erforderlich ist, ist daraufhin unter „Android>Export Android Application“ zu finden (Android.com M).

Der nächste Schritt besteht in der Versionierung. Sie ist ebenfalls nur für den Entwickler beziehungsweise Nutzer zum Beispiel aus Gründen der Kompatibilität relevant und hat ebenfalls keinen Einfluss auf die Installation (Android.com N). Die Informationen über die Version einer App werden in der Datei AndroidManifest.xml, die jedes Android-Projekt beinhaltet, angegeben. Zum einen steht das Attribut android:versionCode in keinem Zusammenhang mit der Version, die dem Nutzer angezeigt wird. Es dient lediglich dazu, den aktuellen Entwicklungsstand einer App festzustellen und sollte mit dem Wert eins beginnend bei jeder weiteren Veröffentlichung monoton erhöht werden. Zum anderen wird durch das Attribut android:versionName dem Nutzer die Version der App zum Beispiel im Android Market angezeigt (Android.com O).

Schließlich kann die App nach der in Kapitel 2.3 erwähnten Registrierung und der Angabe folgender Informationen unter „http://market.android.com/publish“ hochgeladen und veröffentlicht werden:

- Mindestens zwei Screenshots
- Ein hochauflösendes Icon für die Darstellung im Android Market
- Sprache
- Name
- Beschreibung
- App-Typ
- Kategorie
- Bewertung des Inhalts
- Preis

In Abbildung 12 ist schließlich die Veröffentlichung von RolliApp abgeschlossen:

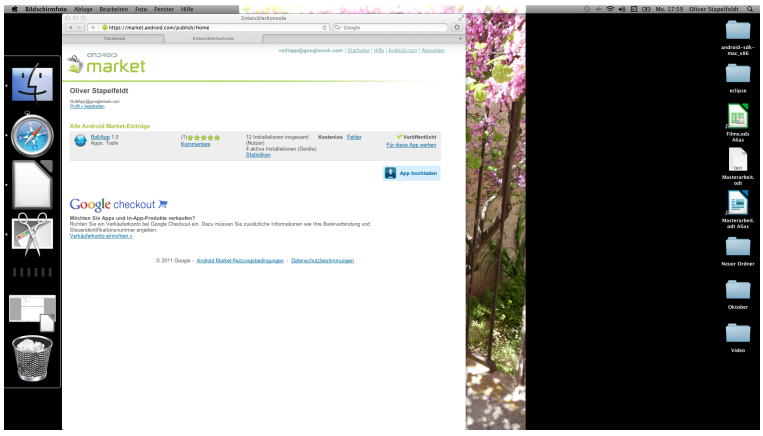

Abbildung 12: RolliApp veröffentlicht

4 Evaluation

Um schließlich die Umsetzung der streckenbezogenen Funktionen auf eine korrekte Funktionsweise hin zu verifizieren, wurde eine Versuchsreihe durchgeführt. Dabei dienten die Stationszeichen, die an Bundes- und Landesstraßen jeweils im Abstand von 200 Metern aufgestellt sind, als Messpunkte, zwischen denen der Abstand mit Hilfe von RolliApp gemessen wurde. Zusätzlich wurde die Abweichung berechnet. Die Versuche wurden lediglich mit einem Testgerät durchgeführt, da zum einen kein weiteres zur Verfügung stand und zum anderen die Ungenauigkeit des GPS-Signals ohnehin entweder von den Satelliten oder dem GPS-Empfänger ausgeht und immer unbeeinflussbar auftreten wird.

Versuch	**Tatsächliche Strecke in m**	**Gemessene Strecke in m**	**Abweichung in %**
1	200	211	5,5
2	200	198	1
3	200	203	1,5
4	200	199	0,5
5	200	198	1
6	200	195	2,5
7	200	198	1
8	200	204	2
9	200	202	1
10	200	204	2
Durchschnittliche Abweichung			1,8

Tabelle 3: Evaluation der streckenbezogenen Funktionen

5 Fazit

Das Ziel, das sich der Autor dieser Masterarbeit gesetzt hatte, konnte mit Hilfe des entwickelten Prototyps erreichet werden. Fahrer von elektrischen Rollstühlen ist es nun grundlegend möglich festzustellen, welche, zuvor selbst definierte, Reichweite noch zurückgelegt werden kann.

Die Berechnung der verbleibenden Reichweite erfolgt beim für das Smartphone-Betriebssystem Android entwickelten Prototyp durch GPS und kann, wie in Kapitel 6 beschrieben, zukünftig um eine Kalibrierung erweitert werden. Der Prototyp wurde sowohl im Emulator innerhalb der Entwicklungsumgebung Eclipse als auch anschließend auf dem Smartphone des Autors unter realen Bedingungen erfolgreich getestet (siehe Kapitel 4). Schließlich wurde RolliApp im Android Market veröffentlicht und kann von jedem Nutzer, der die entsprechenden Voraussetzungen erfüllt, kostenlos heruntergeladen werden.

6 Ausblick

Wie bereits in Kapitel 2.1 erwähnt wurde, bietet RolliApp derzeit keine Möglichkeit, die Alterung des Akkus des elektrischen Rollstuhls zu registrieren. Der Nutzer muss vielmehr die Einstellung für die verbleibende Reichweite manuell anpassen.

Ein möglicher Lösungsansatz könnte so aussehen, dass die App die Anpassung für den Nutzer übernimmt. Der Vorgang könnte beispielsweise beim Zurücksetzen der verbleibenden Reichweite erfolgen. Zusätzlich müsste jedoch ein Schwellenwert von beispielsweise drei Kilometern angegeben werden, damit die Anpassung nicht auch bei aus Vorsicht zurückgesetzter verbleibender Reichweite stattfindet. Weiterhin sollte der Nutzer den Vorgängen zustimmen.

Literaturverzeichnis

androidsmartphone.de

androidsmartphone.de: Google stellt Android 3.0 und Android Market Browser vor. http://androidsmartphone.de/news/google-stellt-android-3-0-und-android-market-browser-vor/ (13.09.2011)

Android.com A

Android.com: Installing the SDK | Android Developers. http://developer.android.com/sdk/installing.html#Preparing (26.08.2011)

Android.com B

Android.com: Installing the SDK | Android Developers. http://developer.android.com/sdk/installing.html#InstallingADT (26.08.2011)

Android.com C

Android.com: ADT Plugin for Eclipse | Android Developers. http://developer.android.com/sdk/eclipse-adt.html#installing (26.08.2011)

Android.com D

Android.com: Installing the SDK | Android Developers. http://developer.android.com/sdk/installing.html#which (26.08.2011)

Android.com E

Android.com: Platform Versions | Android Developers. http://developer.android.com/resources/dashboard/platform-versions.html (29.08.2011)

Android.com F

Android.com: Using the Android Emulator | Android Developers. http://developer.android.com/guide/developing/devices/emulator.html (29.08.2011)

Android.com G

Android.com: Preparing to Publish: A Checklist | Android Developers. http://developer.android.com/guide/publishing/preparing.html#releaseready (29.08.2011)

Android.com H

Android.com: User Interface | Android Developers. http://developer.android.com/guide/topics/ui/index.html#Layout (26.09.2011)

Android.com I

Android.com: More Resource Types | Android Developers. http://developer.android.com/guide/topics/resources/more-resources.html#Dimension (23.09.2011)

Android.com J

Android.com: Icon Design Guidelines | Android Developers. http://developer.android.com/guide/practices/ui_guidelines/icon_design.html#design-tips (28.09.2011)

Android.com K

Android.com: Launcher Icons | Android Developers. http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html#size5 (28.09.2011)

Android.com L

Android.com: Signing Your Applications | Android Developers. http://developer.android.com/guide/publishing/app-signing.html#overview (21.10.2011)

Android.com M

Android.com: Signing Your Applications | Android Developers. http://developer.android.com/guide/publishing/app-signing.html#ExportWizard (24.10.2011)

Android.com N

Android.com: Versioning Your Applications | Android Developers. http://developer.android.com/guide/publishing/versioning.html (24.10.2011)

Android.com O

Android.com: Versioning Your Applications | Android Developers. http://developer.android.com/guide/publishing/versioning.html#appversioning (24.10.2011)

BIG SCREEN Internetportal

BIG SCREEN Internetportal: US-Smartphone-Markt - Android wächst auf 52 Prozent: Google Android baut Führungsposition in den USA weiter aus. http://www.big-screen.de/deutsch/pages/news/allgemeine-news/2011_08_24_7219_us-smartphone-markt-android-waechst-auf-52-prozent.php (24.08.2011)

DalvikVM.com

DalvikVM.com: DalvikVM.com - Dalvik Virtual Machine insights. http://www.dalvikvm.com/#Introduction (25.08.2011)

dauerstress.de

dauerstress.de: HEX-Farbcode – Farbtabelle. http://www.dauerstress.de/homepagehilfe/farbcode.php (23.09.2011)

FIT für Usability. Konzeption, Design & WebBuilding by Peter Hunkirchen.

FIT für Usability. Konzeption, Design & WebBuilding by Peter Hunkirchen.: Kontrast zwischen Text und Hintergrund « Archiv « FIT für Usability. http://www.fit-fuer-usability.de/archiv/kontrast-zwischen-text-und-hintergrund/ (12.09.2011)

Fotolia

Fotolia: Foto: handicap@icon11 (copyright) THesIMPLIFY #32973123. http://de.fotolia.com/id/32973123 (05.08.2011)

Galileo Press

Galileo Press: Galileo Computing :: Java ist auch eine Insel – 1.3 Eigenschaften von Java. http://openbook.galileocomputing.de/javainsel/javainsel_01_003.htm#mjaa295d8b83d9673a3f42c0cb976e3a42 (25.08.2011)

Google A

Google: Developer Registration - Android Market for Developer Help. http://www.google.com/support/androidmarket/developer/bin/answer.py?hl=en&answer=113468 (13.09.2011)

Google B

Google: Kosten - Android Market-Hilfe. http://www.google.com/support/androidmarket/bin/answer.py?hl=de&answer=113408&topic=1100168 (13.09.2011)

Google C

Google: Android is now available as open source (Android Open Source Project). http://web.archive.org/web/20090228170042/http://source.android.com/posts/opensource (24.08.2011)

Köhne; Wößner

Köhne, Anja; Wößner, Michael: Die geschichtliche Entwicklung des GPS-Systems.
http://www.kowoma.de/gps/Geschichte.htm (23.08.2011)

Lichtkreis.at - Spirituelle Wegbegleitung durch Information, Wissen und Lichtarbeit

Lichtkreis.at - Spirituelle Wegbegleitung durch Information, Wissen und Lichtarbeit: Farbe Orange – Wirkung und Bedeutung der Farbe Orange.
http://www.lichtkreis.at/html/Wissenswelten/Welt_der_Farben/wirkung-farbe-orange.htm (12.09.2011)

Open Handset Alliance A

Open Handset Alliance: Alliance Members | Open Handset Alliance.
http://www.openhandsetalliance.com/oha_members.html (24.08.2011)

Open Handset Alliance B

Open Handset Alliance: Android Overview | Open Handset Alliance.
http://www.openhandsetalliance.com/android_overview.html (24.08.2011)

runtastic GmbH A

runtastic GmbH: Impressum.
http://www.runtastic.com/de/impressum (15.09.2011)

runtastic GmbH B

runtastic GmbH: runtastic Features auf den verschiedenen Plattformen.
http://www.runtastic.com/de/mobile-apps/runtastic/featurematrix (15.09.2011)

SELFHTML e.V.

SELFHTML e.V.: SELFHTML: XML / Einführung in XML.
http://de.selfhtml.org/xml/intro.htm#datenfreiheit (26.09.2011)

Anhangsverzeichnis

Auf dem beigefügten Datenträger befindet sich neben der digitalen Version dieser Masterarbeit (Stapelfeldt_Oliver_111123.pdf) das Eclipse-Projekt. Nach dem Dekomprimieren der Datei RolliApp.zip kann das komplette Projekt entweder wie in Kapitel 2.5 beschrieben importiert oder die relevanten Dateien mit Hilfe eines Texteditors betrachtet werden.

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangabe kenntlich gemacht.

Oliver Stapelfeldt

Matrikelnummer: 18 58 50 2