Universität Zürich[UZH]

Simulation einer Elektromotorradfahrt: Modellbildung und Implementierung in Android

Bachelorarbeit im Fach Informatik
vorgelegt von

Marco Creola

Horgen, Schweiz
Matrikel-Nr. 99-708-174

Abgabe der Arbeit

31. Oktober 2011

Supervisor

Prof. Dr. Lorenz M. Hilty
Institut für Informatik der Universität Zürich,
Informatics and Sustainability Research
(http://www.ifi.uzh.ch/departments/isr.html)

Abstract

Die Empa St. Gallen hat eine Sensorenbox entwickelt, die es ermöglicht, während der Fahrt mit einem Elektromotorrad Daten zu erheben und zu sammeln. Die Daten bestehen aus GPS-Werten und Messungen an der Batterie des Elektromotors. Die Fahrt eines Motorrades auf einer geradlinigen Fahrbahn wird von verschiedenen physikalischen Kräften beeinflusst, die lauten: der Luftwiderstand, der Radwiderstand, der Steigungswiderstand, der Beschleunigungswiderstand und die Antriebskraft. Mit Hilfe eines Simulationsmodells, das auf dem System Dynamics Ansatz beruht, werden die physikalischen Kräfte und deren Ursachen, Wirkungen und Rückkoppelungen aufgezeigt und modelliert. Anhand des Simulationsmodells wird eine Applikation basierend auf dem Android Framework entwickelt. Die Applikation ermöglicht einerseits die Verarbeitung der Sensorendaten und die Berechnung der physikalischen Kräfte. Andererseits kann die Fahrt mit anderen Parametern simuliert werden und mit der tatsächlichen Fahrt verglichen werden.

Danksagung

Ohne die Hilfe einiger Personen wäre es nicht möglich gewesen, diese Arbeit so zu erstellen. Ich möchte es nicht versäumen, mich bei ihnen auf diesem Weg zu bedanken.

Ich danke Herrn Prof. Dr. Lorenz M. Hilty für die Unterstützung und die Möglichkeit, die Arbeit zu diesem Thema erstellen zu dürfen. Ebenso danke ich seinen Mitarbeitern von der Empa St. Gallen, Herrn Rolf Widmer und Herrn Marcel Gauch, für ihren Input und ihre Hilfe.

Den MitarbeiterInnen meiner Arbeitgeberin, iET SA, danke ich für das Verständnis und das Entgegenkommen bei der Einteilung meiner Arbeitszeit.

Und schliesslich möchte ich mich bei Carla Loretz recht herzlich für all ihre Unterstützung bedanken.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den ersten sechs Monaten dieses Jahres wurden in Europa doppelt so viele Elektromotorräder verkauft wie im letzten Jahr im gleichen Zeitraum (ACEM, 2011). Im Jahr 2010 wurden in der Schweiz vierzigmal so viele Elektromotorräder wie im Jahre 2002 verkauft (eco-way, 2011). Die österreichische Niederlassung des Fraunhofers Institut führte gemäss einem diesjährigen Report eine Marktanalyse durch, die zum Schluss kam, dass sich Elektromobilität – zumindest in der Automobilindustrie – langfristig durchsetzen wird (Fraunhofer Austria, 2011). Der Trend zu mehr Elektromobilität scheint unbestritten zu sein. Um die von der Schweiz und anderen Staaten in den nächsten Jahren angestrebte Reduktion der CO_2-Emissionen erreichen zu können, geht das Deutsche Zentrum für Luft- und Raumfahrt davon aus, dass dies ohne den Wechsel hin zu mehr Elektromobilität nicht machbar sei (Parlamentsdienste, 2011; DLR, 2011). Ob und wie die Elektromobilität zu den Reduktionszielen zukünftig beitragen kann und soll, wird sich zeigen.

An der Eidgenössischen Materialprüfungs- und Forschungsanstalt (Empa) beschäftigt man sich schon länger mit der Elektromobilität. In diesem Zusammenhang wurde an der Empa St. Gallen eine Sensorenbox entwickelt, die es ermöglicht, während der Fahrt mit einem Elektromotorrad Daten zu messen und zu sammeln. Einerseits wird mit Hilfe eines GPS-Gerätes Positionssignale von Satelliten empfangen und ausgewertet. Andererseits werden Messungen an der Batterie, die den Elektromotor mit Energie versorgt, durchgeführt.

Auf ein Motorrad – egal, ob elektrisch oder durch fossile Energieträger angetrieben – wirken physikalische Kräfte, die überwunden werden müssen, um das Motorrad in Bewegung zu setzen bzw. in Bewegung zu halten. Die vorliegende Arbeit ergänzt die Entwicklung der Sensorenbox, indem die gesammelten Daten und die auf ein Motorrad wirkenden Kräfte mit Hilfe von Software verarbeitet werden. Anhand der Implementierung eines Modelles sollen Simulationen auf Basis der gesammelten Daten und den physikalischen Kräften durchgeführt und ausgewertet werden können.

Im ersten Teil wird das erarbeitete Modell beschrieben. Dazu wird das Modell zuerst abgegrenzt und getroffene Annahmen beschrieben. Danach werden alle physikalischen Kräfte, die auf ein Motorrad wirken, erarbeitet. Schliesslich werden die physikalischen Kräfte in einem Modell konsolidiert und grafisch dargestellt. Die Modellbildung beruht auf dem System Dynamics Ansatz, welcher zuvor beschrieben wird.

Im zweiten Teil wird die Softwareimplementierung des erarbeiteten Modelles aufgezeigt. Es wurde entschieden, dass das Simulationsmodell auf einer Plattform entwickelt werden soll, das auf mobilen Geräten zum Einsatz kommt. Deshalb wurde für die Entwicklung der Applikation das Android Framework gewählt. Über die Plattform und deren Entwicklungsumgebung wird eine kurze Einführung gegeben. Danach werden das Konzept und die benötigten Inputdaten der Sensorenbox beschrieben. Schliesslich werden alle Anwendungsfälle mit detaillierten Funktionsbeschreibungen aufgezeigt und erläutert. Die Beschreibungen zum verwendeten Datenmodell und der gewählten Datenbankimplementierung schliessen das Kapitel ab.

Kapitel 2

Das Modell

Dieses Kapitel beschreibt das Modell, das zur Simulation der auf ein Elektromotorrad wirkenden Kräfte benötigt wird.

Im ersten Teil wird das Modell abgegrenzt und Annahmen werden getroffen. Danach werden die physikalischen Kräfte und Widerstände, die auf ein Motorrad wirken, beschrieben. Der letzte Teil des Kapitels zeigt schliesslich, wie diese Kräfte in einem Simulationsmodell repräsentiert werden.

2.1 Abgrenzung und Annahmen

Das zu entwickelnde Modell soll das reale Gebilde eines Motorrades und dessen Fahrt simulieren. Insbesondere gilt es, die verschiedenen Kräfte, die auf ein Motorrad wirken, zu modellieren. Das Modell wird folgendermassen vereinfacht: das Motorrad wird als bewegter Massepunkt auf einer geradlinigen Fahrbahn betrachtet. Das heisst erstens, dass Auswirkungen einer möglichen ungleich verteilten Masse auf die beiden Räder nicht berücksichtigt werden. Zweitens werden Kräfte, die in Kurven auf das Motorrad wirken (Vittore, 2006), ebenfalls nicht berücksichtigt.

Annahmen und Abgrenzungen, die spezifisch auf die wirkenden Kräfte gelten, werden in den nachfolgenden Abschnitten jeweils separat erwähnt.

Abbildung 2.1 zeigt das abstrahierte Modell, auf das sich die nachfolgenden Beschreibungen beziehen.

2.2 Physikalische Kräfte

Die verschiedenen Kräfte, die auf ein Motorrad bei geradliniger Fahrt wirken, sind: der *Luftwiderstand*, der *Radwiderstand*, der *Steigungswiderstand*, der *Beschleunigungswiderstand* und die *Antriebskraft* (Vittore (2006); Mitschke und Wallentowitz (2004); siehe auch Abbildung 2.1).

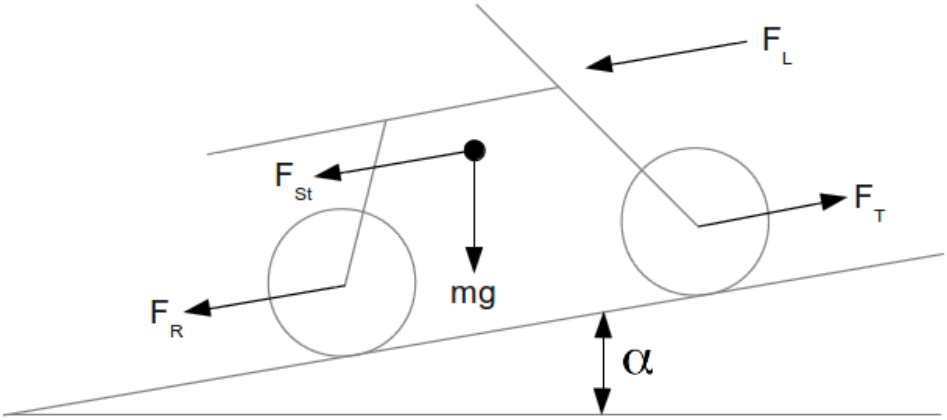


Abbildung 2.1: Modell der wirkenden Kräfte (vgl. Vittore (2006))

Physikalische Kräfte werden in der Masseinheit Newton (N) gemessen (Bader und Dorn, 1996):

$$1\ \mathrm{N} = 1\ \frac{\mathrm{kg\ m}}{\mathrm{s}^2} \tag{2.1}$$

Die erwähnten Kräfte werden in den folgenden Unterabschnitten einzeln beschrieben.

2.2.1 Luftwiderstand

Sich bewegende Körper erfahren Strömungswiderstände durch das sie umgebende Medium (Flüssigkeit oder Gas) (Heissing et al., 2011; Bader und Dorn, 1996). Um den Bewegungszustand aufrecht zu erhalten, müssen diese Strömungswiderstände überwunden werden (Heissing et al., 2011). Bei einem fahrenden Motorrad ist das umgebende Medium die Luft, welches zu überwinden gilt. In Abbildung 2.1 wird der Luftwiderstand als F_L dargestellt.

Die mathematische Gleichung für den Luftwiderstand lautet (Vittore, 2006; Mitschke und Wallentowitz, 2004; Bader und Dorn, 1996; Heissing et al., 2011):

$$F_L = p * c_L * A \begin{cases} F_L: & \text{Luftwiderstandskraft (N)} \\ p: & \text{Staudruck (N/m}^2\text{)} \\ c_L: & \text{Dimensionsloser Luftwiderstandskoeffizient} \\ A: & \text{Stirnfläche (m}^2\text{)} \end{cases} \tag{2.2}$$

Die einzelnen Faktoren werden nun im Detail besprochen.

Der Staudruck p lässt sich anhand der Luftdichte ρ_L (kg/m^3) und der Relativ- bzw. Anströmungsgeschwindigkeit v_{rel} (m/s) berechnen (Heissing et al., 2011):

$$p = \frac{\rho_L}{2} * v_{rel}^2 \tag{2.3}$$

Die Gleichung (2.2) lässt sich deshalb auch so formulieren:

$$F_L = \frac{\rho_L}{2} * v_{rel}^2 * c_L * A \tag{2.4}$$

Die Luftdichte ρ_L hängt von verschiedenen Werten ab: vom Luftdruck ρ_U, von der Umgebungstemperatur Θ_U und von der Gaskonstante R_L (Heissing et al., 2011):

$$\rho_L = \frac{\rho_U}{R_L * \Theta_U} \tag{2.5}$$

Die Gaskonstante wiederum hängt von der Luftfeuchtigkeit ab (Ruppelt, 2003).

Die Relativgeschwindigkeit v_{rel} setzt sich aus der Fahrgeschwindigkeit des Motorrades und der Windgeschwindigkeit (Rücken- oder Gegenwind) zusammen (Stoffregen, 2010):

$$v_{rel} = v_{Motorrad} \pm v_{Wind} \tag{2.6}$$

Der Luftwiderstand wird demnach von aktuellen Umweltbedingungen (Luftdruck, Umgebungstemperatur, Luftfeuchtigkeit und Windstärke) beeinflusst.

Wie in Kapitel 3 beschrieben, stehen der Simulation die Werte des Luftdrucks, der Temperatur und der Gaskonstante (bzw. der Luftfeuchtigkeit) nicht zur Verfügung. Somit kann kein exakter Wert für die Luftdichte berechnet werden. Deshalb wird für die Luftdichte ein Standardwert eingesetzt, der sich bei trockener Luft, bei 0° C und dem Normdruck ergibt (DMK und DPK, 1995):

$$\rho_L = 1.293 kg/m_3 \tag{2.7}$$

Der dimensionslose Luftwiderstandskoeffizient c_L „wird experimentell für jedes Fahrzeug im Windkanal bestimmt“ (Heissing et al., 2011). Der Luftwiderstandskoeffizient c_L bildet einerseits die Form- und Strömungsgüte des Fahrzeuges ab, andererseits aber auch die Windschlüpfrigkeit des Fahrers/der Fahrerin (Stoffregen, 2010). Ob der Fahrer/die Fahrerin auf dem Motorrad aufrecht sitzt oder liegt, beeinflusst den Luftwiderstandskoeffizienten (Stoffregen, 2010). Kurz: je windschlüpfiger ein Motorrad und dessen Fahrer/deren Fahrerin (bzw. dessen/deren Fahrposition) sind, desto kleiner ist der Luftwiderstandskoeffizient (Stoffregen, 2010; Heissing et al., 2011).

Die Stirnflächengrösse A entspricht dem flächenmässig grössten Querschnitt des Motorrades und des Fahrers/der Fahrerin senkrecht zur Strömung (Stoffregen, 2010). Auch hier wiederum spielt die Sitzposition des Fahrers/der Fahrerin eine Rolle.

Zusammenfassend kann man festhalten, dass der Luftwiderstand quadratisch zur Relativgeschwindigkeit steigt und von aktuellen Umweltbedingungen, der Strömungsgüte und der Stirnfläche des Fahrzeuges abhängt.

2.2.2 Radwiderstand

Der Radwiderstand (siehe F_R in Abbildung 2.1) bei geradliniger Fahrt ist von mehreren Faktoren abhängig: Gewicht des Motorrades (inkl. Ladung), Fahrbahnsteigung, Reifenmaterial, Beschaffenheit der Fahrbahn und Lagerreibung (Haken, 2008; Mitschke und Wallentowitz, 2004). Der Radwiderstand wird für diese Arbeit soweit vereinfacht, sodass nur das Gewicht des Motorrades, die Fahrbahnsteigung und das Reifenmaterial berücksichtigt werden.

Für den Radwiderstand F_R gilt folgende Gleichung (Haken, 2008; Mitschke und Wallentowitz, 2004):

$$F_R = f_R * F_N \begin{cases} F_R: & \text{Radwiderstandskraft (N)} \\ f_R: & \text{Dimensionsloser Rollwiderstandskoeffizient} \\ F_N: & \text{Normalkraft (N)} \end{cases} \tag{2.8}$$

Der dimensionslose Rollwiderstandsbeiwert f_R repräsentiert die Beschaffenheit des Reifens, d.h. Reifendruck, -material, -temperatur und die Verformung des Reifens an der Bodenkontaktfläche (Haken, 2008; Mitschke und Wallentowitz, 2004). Der Rollwiderstandsbeiwert f_R wird bei Laborversuchen und/oder Strassentests ermittelt (Drews et al., 1991; Hucho, 2008). Zahlreiche Messungen an Motorrädern haben ergeben, dass sich auf trockener Fahrbahn ein durchschnittlicher Rollwiderstandsbeiwert von

$$f_R = 0.02 \tag{2.9}$$

einstellt (Vittore, 2006).

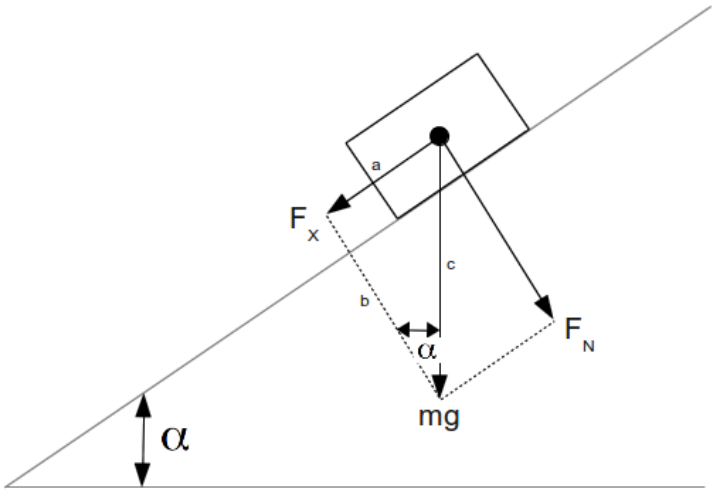


Abbildung 2.2: Normal- und Gewichtskraft (vgl. Bader und Dorn (1996))

Abbildung 2.2 zeigt ein abstrahiertes Modell eines Körpers in einer Steigung und die darauf wirkende Normalkraft F_N, die Gewichtskraft mg und der Rollwiderstand (in Abbildung

2.2 als F_X dargestellt). Die Normalkraft F_N lässt sich anhand des Steigungswinkels α, der Gewichtskraft und des Kosinussatzes[1] errechnen (Bader und Dorn, 1996):

$$F_N = mg * cos(\alpha) \tag{2.10}$$

Setzt man nun die Gleichung (2.10) in die Gleichung (2.8) ein, ergibt sich folgende vollständige Gleichung des Radwiderstandes:

$$F_R = (m_{Moto} + m_{Lad}) * g * f_R * cos(\alpha) \tag{2.11}$$

Die Variable m_{Moto} steht dabei für das Gewicht des Motorrades (kg). Analog dazu steht die Variable m_{Lad} für das Gewicht der Ladung (inkl. FahrerIn).

2.2.3 Steigungswiderstand

Der Steigungswiderstand (siehe F_{St} in Abbildung 2.1) setzt sich aus der Schwerkraft und dem Steigungswinkel zusammen (Stoffregen, 2010; Vittore, 2006). Bergauf nimmt der Steigungswiderstand positive, bergab negative Werte an und bei horizontaler Fahrt ist der Steigungswiderstand gleich 0 (Stoffregen, 2010). Bei einem Gefälle wird er so zu einer zusätzlichen Antriebskraft (Heissing et al., 2011). Das bedeutet, dass bei einer Fahrt mit geografisch identischem Start- und Zielpunkt der Steigungswiderstand in seiner Summe gleich 0 ist (Heissing et al., 2011).

Die mathematische Gleichung für den Steigungswiderstand lautet (Vittore, 2006; Mitschke und Wallentowitz, 2004; Bader und Dorn, 1996; Heissing et al., 2011):

$$F_{St} = (m_{Moto} + m_{Lad}) * g * sin(\alpha) \begin{cases} F_{St}: & \text{Steigungswiderstandskraft (N)} \\ m_{Moto}: & \text{Gewicht des Motorrades (kg)} \\ m_{Lad}: & \text{Gewicht der Ladung (kg)} \\ g: & \text{Erdbeschleunigung (m/s}^2\text{)} \\ \alpha: & \text{Steigungswinkel (rad)} \end{cases} \tag{2.12}$$

Die Gleichung lässt sich auch anhand der Abbildung 2.2 ableiten. Mittels des Sinussatzes[2] lässt sich der Steigungswiderstand (in Abbildung 2.2 als F_X dargestellt) mit Hilfe der Gewichtskraft mg und dem Steigungswinkel α errechnen (Bader und Dorn, 1996).

2.2.4 Beschleunigungswiderstand

Bei einer Änderung des Bewegungszustandes (Richtung bzw. Geschwindigkeit) eines Körpers muss der Trägheits- bzw. Beschleunigungswiderstand F_A überwunden werden (Stoffregen, 2010; Heissing et al., 2011; Bader und Dorn, 1996). Eine Änderung des Bewegungszustandes tritt dann also ein, wenn der Körper eine positive oder negative Beschleunigung

[1] $cos(\alpha) = \frac{\text{Ankathete (b)}}{\text{Hypotenuse (c)}}$ (DMK und DPK, 1995).
[2] $sin(\alpha) = \frac{\text{Gegenkathete (a)}}{\text{Hypotenuse (c)}}$ (DMK und DPK, 1995).

erfährt[3], d.h. wenn folgendes gilt (Heissing et al., 2011):

$$a = \frac{v_2 - v_1}{t_2 - t_1} \neq 0 \tag{2.13}$$

Dabei wird zwischen dem *translatorischen* und dem *rotatorischen Widerstand* unterschieden (Stoffregen, 2010). Die Widerstände werden addiert, um den gesamten Beschleunigungswiderstand zu erhalten (Stoffregen, 2010).

Der *translatorische Widerstand* wirkt bei der Beschleunigung des gesamten Motorrades (Stoffregen, 2010). Die mathematische Gleichung dazu sieht wie folgt aus:

$$F_{A,trans} = (m_{Moto} + m_{Lad}) * a \begin{cases} F_{A,trans}: & \text{Translatorische Widerstandskraft (N)} \\ m_{Moto}: & \text{Gewicht des Motorrades (kg)} \\ m_{Lad}: & \text{Gewicht der Ladung (kg)} \\ a: & \text{Beschleunigung (m/s}^2\text{)} \end{cases} \tag{2.14}$$

Die Gleichung 2.14 entspricht dem Newtonschen Beschleunigungssatz ($F = ma$), der die Grundgleichung der Mechanik bildet (Bader und Dorn, 1996).

Der *rotatorische Widerstand* wirkt bei der Beschleunigung der drehenden Teile (Motor, Getriebe und Räder) des Antriebsstranges (Stoffregen, 2010; Heissing et al., 2011). Der rotatorische Widerstand wirkt nur auf das Motorrad selber und nicht auf die Ladung (d.h. FahrerIn und sonstige Ladung) (Heissing et al., 2011). Stoffregen (2010) hält fest, dass „die Drehmomentberechnung für jedes Bauteil einzeln vorgenommen werden“ muss, „da die drehenden Teile von Motor und Antriebsstrang unterschiedliche Massenträgheitsmomente“ aufweisen. Der rotatorische Beschleunigungswiderstand $F_{A,rot}$, der am Antriebsrad überwindet werden muss, errechnet sich wie folgt (Heissing et al., 2011):

$$F_{A,rot} = \frac{\Theta_{red}}{r_{dyn}^2} * a \begin{cases} F_{A,rot}: & \text{Rotatorische Widerstandskraft (N)} \\ \Theta_{red}: & \text{Massenträgheitsmoment (kgm}^2\text{)} \\ r_{dyn}: & \text{Dynamischer Radhalbmesser}^4 \text{ (m)} \\ a: & \text{Beschleunigung (m/s}^2\text{)} \end{cases} \tag{2.15}$$

Die Addierung der Gleichung (2.14) und (2.15) ergibt den gesamten Beschleunigungswiderstand:

$$F_A = \left((m_{Moto} + m_{Lad}) + \frac{\Theta_{red}}{r_{dyn}^2} \right) * a \tag{2.16}$$

Durch die Einführung eines Massenfaktors e (2.17) kann die Gleichung (2.16) vereinfacht werden (Heissing et al., 2011):

$$e = \frac{\Theta_{red}}{m_{Moto} * r_{dyn}^2} + 1 \tag{2.17}$$

$$F_A = (e * m_{Moto} + m_{Lad}) * a \tag{2.18}$$

[3]Wie in Abschnitt 2.1 erwähnt, wird das Modell vereinfacht und nur die geradlinige Fahrt ohne Richtungswechsel betrachtet.

[4]Radius des Rades anhand des tatsächlichen Abrollumfangs (Haken, 2008).

2.2.5 Antriebskraft und Antriebsleistung

Wie die vorhergehenden Abschnitte (2.2.1, 2.2.2, 2.2.3 und 2.2.4) gezeigt haben, ergibt sich ein Gesamtwiderstand F_{Total} von:

$$F_{Total} = F_L + F_R + F_{St} + F_A \tag{2.19}$$

Die Antriebskraft F_T am Rad, die es braucht, um die erwähnten Widerstandskräfte zu überwinden, bildet sich anhand der entgegengesetzten Gesamtwiderstandskraft (Heissing et al., 2011):

$$F_T = -F_{Total} \tag{2.20}$$

Da es sich um entgegengesetzte Vektoren handelt, muss F_{Total} mit -1 multipliziert werden (Bader und Dorn, 1996).

Neben der Antriebskraft F_T interessiert vor allem auch die Antriebsleistung P_T, welche erforderlich ist, um die Relativgeschwindigkeit eines Motorrades zu halten. Sie errechnet sich wie folgt (Bader und Dorn, 1996):

$$P_T = F_{Total} * v_{rel} \begin{cases} P_T: & \text{Antriebsleistung (Watt)} \\ F_{Total}: & F_L + F_R + F_{St} + F_A \text{ (N)} \\ v_{rel}: & \text{Relativgeschwindigkeit (m/s)} \end{cases} \tag{2.21}$$

2.3 Das Simulationsmodell

In diesem Abschnitt wird das Simulationsmodell erstellt. Zuerst wird der theoretische Hintergrund zu *System Dynamics* erläutert. Danach werden die aus dem Abschnitt 2.2 beschriebenen Gleichungen in das Simulationsmodell überführt.

2.3.1 Der System Dynamics Ansatz

Die Abbildung des Modells erfolgt aufgrund der Theorie zu *System Dynamics*. Deshalb wird hier kurz darauf eingegangen, was unter System Dynamics verstanden wird.

System Dynamics wurde in den 1950er Jahren von Jay W. Forrester am Massachusetts Institute of Technology (MIT) begründet (System Dynamics Society, 2011). System Dynamics befasst sich damit, wie dynamische Systeme modelliert und simuliert werden können (Schöneborn, 2004).

Roberts et al. (1983) definieren Simulation kurz und prägnant mit: „to simulate is to imitate something“. Um etwas simulieren zu können, braucht es vom Simulationsgegenstand ein entsprechendes Modell (Roberts et al., 1983). Das Modell imitiert während der Simulation die wichtigsten Elemente des realen Simulationsgegenstandes (Roberts et al., 1983). Mit dem System Dynamics Ansatz können einzelne Elemente eines Systems kausal

verknüpft werden (Schöneborn, 2004). Gemäss Schöneborn (2004) können deshalb „dynamische und auch zeitverzögerte Wechselwirkungen“ zwischen Systemelementen abgebildet werden.

Bei System Dynamics spielt der Zusammenhang zwischen Ursache, Wirkung und allfälliges Feedback eine zentrale Rolle (Roberts et al., 1983). Der System Dynamics Ansatz führt dazu eine Reihe von grafischen Abbildungen (Diagrammen) ein (Roberts et al., 1983). Um die Diagramme zu erklären, wird ein konkretes Beispiel genommen, dass mit dem System Dynamics Ansatz abgebildet werden kann und die elementarsten Diagrammtypen erklärt: die Beeinflussung der Geburten auf eine Population.

Abbildung 2.3 zeigt anhand des Beispiels die Darstellung von Ursache und Wirkung im System Dynamics Ansatz. Das Diagramm wird anhand des Richtungspfeils folgendermassen gelesen: die Geburten beeinflussen die Population (Roberts et al., 1983).

Abbildung 2.3: Ursache und Wirkung I (vgl. Roberts et al. (1983))

Will man zusätzlich eine Aussage darüber machen, ob eine Beeinflussung positiv oder negativ ist, werden die Richtungspfeile mit einem „+“ oder einem „-“ versehen (Roberts et al., 1983). Abbildung 2.4 wird folgendermassen gelesen: je mehr Geburten es gibt, desto grösser wird die Population (Roberts et al., 1983). Abbildung 2.4 heisst aber auch: je weniger Geburten es gibt, desto kleiner wird die Population (Roberts et al., 1983). Wäre der Pfeil in Abbildung 2.4 mit einem „-“ versehen, würde die Aussage folgendermassen lauten: je mehr Geburten es gibt, desto kleiner wird die Population bzw. je weniger Geburten es gibt, desto grösser wird die Population (Roberts et al., 1983).

Abbildung 2.4: Ursache und Wirkung II (vgl. Roberts et al. (1983))

Wie bereits erwähnt, spielen Wechselwirkungen (Feedbacks) beim System Dynamics Ansatz eine wichtige Rolle (Schöneborn, 2004). Das bereits benutzte Beispiel liefert ein solches Feedback: die Populationsgrösse beeinflusst nämlich wiederum die Geburten. Abbildung 2.5 zeigt eine solche Wechselwirkung. Das Diagramm wird folgendermassen gelesen: je mehr Geburten, desto grösser die Population, und je grösser die Population, desto mehr Geburten gibt es (Roberts et al., 1983). Hierbei handelt es sich um einen sogenannten positiven Feedback Loop: es besteht einerseits eine Wechselwirkung zwischen Ursache und Wirkung und andererseits wird diese Wechselwirkung positiv verstärkt (Roberts et al., 1983). Loops werden ebenfalls mit einem „+“ oder einem „-“[5] versehen (siehe dazu den kreisförmigen Pfeil in der Mitte der Abbildung 2.5). Loops können zudem zu immer komplexer werdenden Gebilde erweitert bzw. verfeinert werden (Roberts et al., 1983).

[5]Um zu bestimmen, ob ein Loop positiv oder negativ ist, zählt man alle vorkommenden „-“ Zeichen zusammen. Ist die Anzahl der „-“ Zeichen gerade oder gleich 0, handelt es sich um einen positiven Feedback Loop, ansonsten um einen negativen (Roberts et al., 1983).

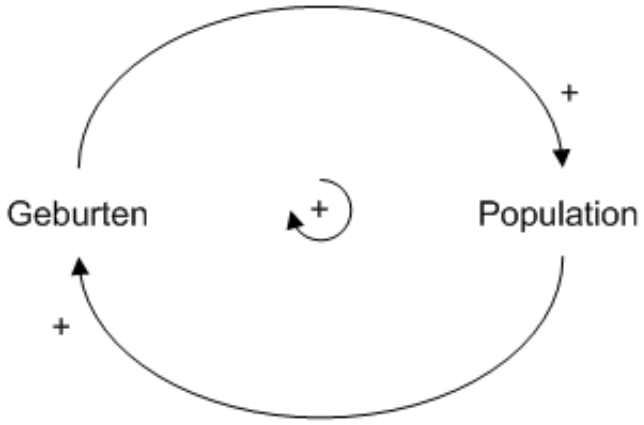


Abbildung 2.5: Causal Loop Diagramm (vgl. Roberts et al. (1983))

Um schliesslich ein Simulationsmodell zu erhalten, wird das Causal Loop Diagramm (auch Wirkungsdiagramm genannt) in ein Flussdiagramm (Stock und Flow Diagramm) transformiert (Schmidt, 2009). Stock und Flow Diagramme ermöglichen einen tieferen Einblick in Feedback Systeme als Causal Loop Diagramme, denn sie identifizieren Levels und Raten (Englisch „Level and Rates“) (Roberts et al., 1983). Ein Level repräsentiert eine Akkumulierung über eine gewisse Zeitspanne (= Stock) (Roberts et al., 1983). Gemessen wird ein Level in Einheiten wie z.B. die Anzahl Menschen (Population) (Roberts et al., 1983). Levels können sich nur durch veränderliche Rates ändern (Roberts et al., 1983). Rates werden in Einheiten pro Zeitspanne gemessen, z.B. die Geburtsrate (Roberts et al., 1983).

Abbildung 2.6 zeigt die Rate und den Level unseres Beispiels. Die Rate „Geburten“ beeinflusst den Level „Population“, was wiederum die Rate beeinflusst (gestrichelte Linie). Die in Abbildung 2.6 gezeigte Wolke bildet dabei die Quelle der Rate ab (Roberts et al., 1983). Quellen stehen stellvertretend für Levels und Rates ausserhalb des betrachteten Systems (Roberts et al., 1983).

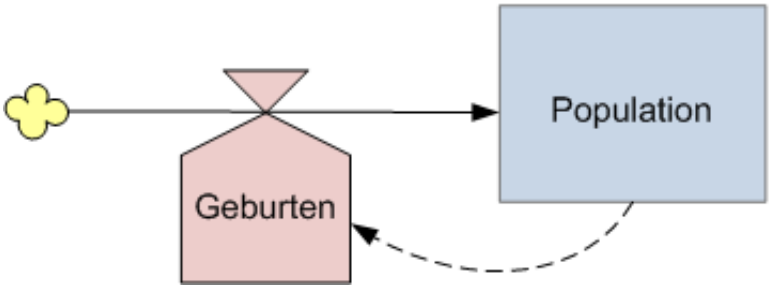


Abbildung 2.6: Stock und Flow Diagramm (vgl. Roberts et al. (1983))

Der Diagrammtyp „Stock und Flow Diagramm“ stellt noch zwei weitere Elemente zur Verfügung: *Konstanten* und *Auxiliaries*. Wie in Abschnitt 2.2 gezeigt wurde, spielt z.B. die Erdbeschleunigungskraft g bei einigen Widerstandskräften eine Rolle. Die Erdbeschleunigungskraft entspricht einem konstanten Wert, der innerhalb des betrachteten Systems unveränderlich ist. Abbildung 2.7 zeigt die Repräsentation einer Konstante innerhalb eines

Stock und Flow Diagramms. Auxiliaries (siehe Abbildung 2.7) werden als Hilfselemente verwendet, die zur Bildung von Rates benötigt werden; also dafür, dass die grafische Repräsentation übersichtlicher wird (Roberts et al., 1983).

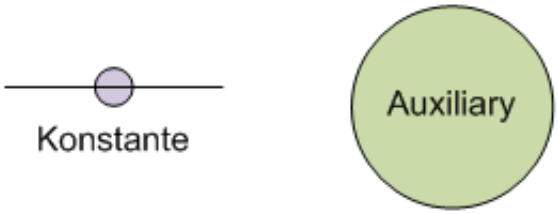


Abbildung 2.7: Konstante und Auxiliary (vgl. Roberts et al. (1983))

Vergleicht man die Abbildungen 2.5 und 2.6 miteinander, zeigt sich, dass anhand des Stock und Flow Diagrammes klarer ist, wie sich die Systemelemente voneinander unterscheiden (Level und Rate) (Roberts et al., 1983). Das wie in Abbildung 2.5 gezeigte Causal Loop Diagramm dient deshalb als Hilfestellung zur Identifikation der Level und Rates, was sich erst bei komplexeren Modellen zeigen wird (Roberts et al., 1983). Folgerichtig ist deshalb, dass bei einer mit dem System Dynamics Ansatz gewählten Systemmodellierung zuerst das vereinfachte Causal Loop Diagramm und erst danach das detaillierte Stock und Flow Diagramm entworfen wird (Roberts et al., 1983).

Nach der Entwicklung des Stock und Flow Diagramms ist der nächste Schritt in der Modellbildung, das Diagramm in mathematische Gleichungen zu überführen (Roberts et al., 1983). Da dieser Schritt für die vorliegende Arbeit nicht relevant ist, wird hier auch nicht darauf eingegangen.

2.3.2 Modellbildung

Wie in Abschnitt 2.3.1 erwähnt, wird bei der Modellbildung mit dem System Dynamics Ansatz zuerst das Causal Loop Diagramm entworfen.

Im Causal Loop Diagramm sollen alle systemrelevanten Elemente abgebildet werden. In Abschnitt 2.2 wurden die Kräfte, die auf ein Motorrad wirken, erwähnt: der Luftwiderstand, der Radwiderstand, der Steigungswiderstand, der Beschleunigungswiderstand und die Antriebskraft. Alle diese Kräfte bzw. deren Elemente, welche die Kraft bilden, sollen nun mit Hilfe eines Causal Loop Diagrammes abgebildet werden. Tabelle 2.1 zeigt alle Kräfte und deren Elemente.

Abbildung 2.8 zeigt das entwickelte Causal Loop Diagramm aufgrund der aufgeführten Elemente in der Tabelle 2.1. Die Beziehungen der einzelnen Elemente werden nun beschrieben.

Der Luftwiderstand F_L wird von der Luftdichte ρ_L, von der Stirnfläche A, von der Relativgeschwindigkeit v_{rel} und vom Luftwiderstandskoeffizient c_L positiv beeinflusst. Das heisst also, je grösser die Werte der vier genannten Elemente sind, desto grösser wird der Luftwiderstand. Dieser wiederum beeinflusst die Beschleunigung des Motorrades a_M

Kraft	**Element**	**Beschreibung**
Luftwiderstand F_L	ρ_L	Luftdichte
siehe Gleichung (2.4)	v_{rel}	Relativgeschwindigkeit
	c_L	Luftwiderstandskoeffizient
	A	Stirnfläche
Radwiderstand F_R	m_{Moto}	Gewicht Motorrad
siehe Gleichung (2.11)	m_{Lad}	Gewicht Ladung
	g	Erdbeschleunigung
	f_R	Rollwiderstandskoeffizient
	α	Steigungswinkel
Steigungswiderstand F_{St}	m_{Moto}	Gewicht Motorrad
siehe Gleichung (2.12)	m_{Lad}	Gewicht Ladung
	g	Erdbeschleunigung
	α	Steigungswinkel
Beschleunigungswiderstand F_A	e	Massenfaktor
siehe Gleichung (2.18)	m_{Moto}	Gewicht Motorrad
	m_{Lad}	Gewicht Ladung
	a	Beschleunigung
Antriebskraft F_T	F_T	Traktionskraft am Rad

Tabelle 2.1: Kräfte und deren Elemente

negativ, d.h. je grösser der Luftwiderstand, desto kleiner wird die Beschleunigung des Motorrades.

Der Radwiderstand F_R wird vom Rollwiderstandskoeffizienten f_R positiv, vom Gesamtgewicht $m_{Moto+Lad}$ positiv, von der Erbeschleunigungskonstante g neutral und vom Steigungswinkel α negativ beeinflusst. Die Erdbeschleunigung beeinflusst den Radwiderstand „neutral“, da es sich um eine unveränderliche Konstante handelt. Der Steigungswinkel wirkt negativ, da gemäss Gleichung (2.11) der Kosinus von α verwendet wird und je grösser α ist, desto kleiner wird der Kosinus von α (Deutsche Mathematiker-Vereinigung, 2011). Wie beim Luftwiderstand gilt auch beim Radwiderstand: je grösser der Widerstand ist, desto kleiner wird die Beschleunigung a_M.

Der Steigungswiderstand F_{St} wird vom Steigungswinkel α positiv, von der Erdbeschleunigungskonstante g neutral und vom Gesamtgewicht $m_{Moto+Lad}$ positiv beeinflusst. Vom Steigungswinkel wird in der Gleichung (2.12) der Sinus verwendet, d.h. je grösser der Winkel α ist, desto grösser ist der Sinus von α (Deutsche Mathematiker-Vereinigung, 2011). Der Steigungswiderstand hat eine negative Wirkung auf die Beschleunigung a_M.

Der Beschleunigungswiderstand F_A wird von der Motorradbeschleunigung a_M, vom Gesamtgewicht $m_{Moto+Lad}$ und vom Massenfaktor e positiv beeinflusst. Der Beschleunigungswiderstand wirkt analog zu den anderen Widerständen negativ auf die Beschleunigung a_M.

Die Antriebskraft F_T hat einen positiven Einfluss auf die Beschleunigung a_M, d.h. je grösser die Traktionskraft am Rad ist, desto grösser wird die Beschleunigung des Motorrades.

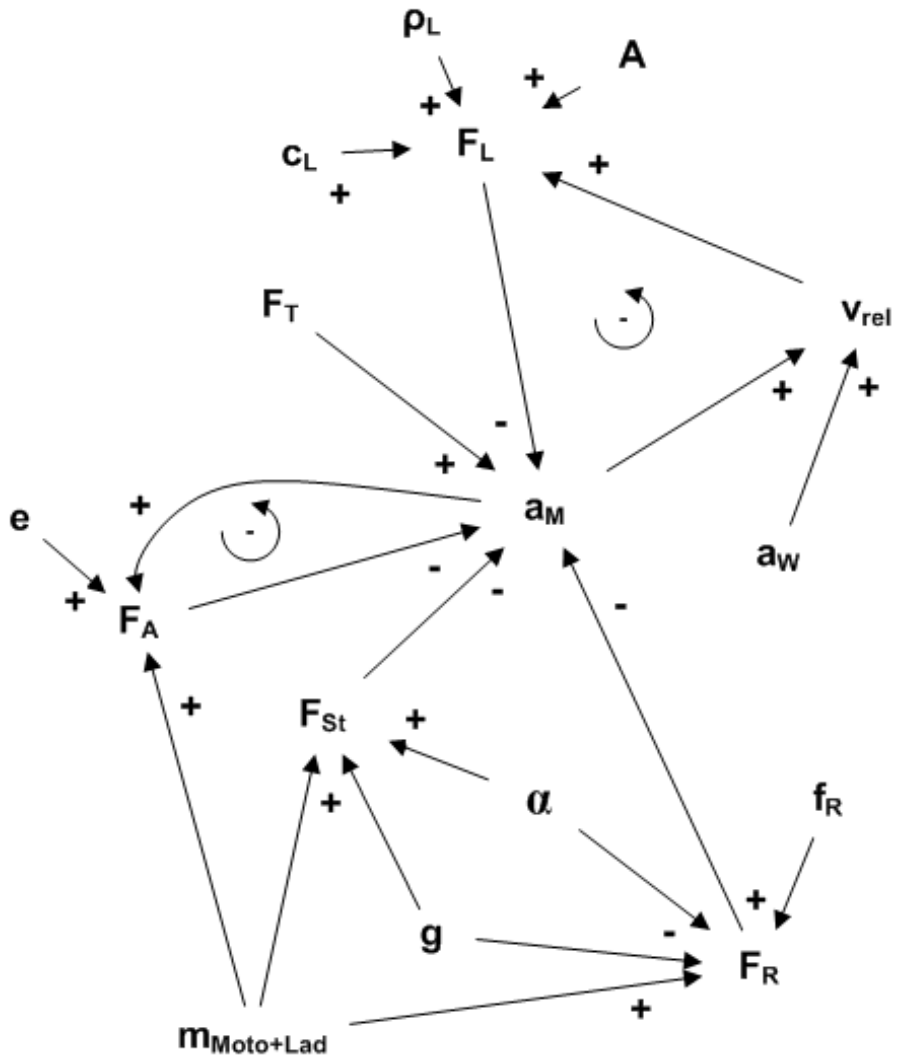


Abbildung 2.8: Causal Loop Diagramm Motorrad

Schliesslich bleibt noch die Beschleunigung durch den Wind a_W (Rücken- oder Gegenwind), die – analog zur Motorradbeschleunigung a_M – die Relativgeschwindigkeit v_{rel} positiv beeinflusst (vgl. Gleichung (2.6)).

Abbildung 2.8 zeigt insgesamt zwei negative Feedback Loops (vgl. Abschnitt 2.3.1). Der erste Feedback Loop betrifft den Luftwiderstand F_L und liest sich folgendermassen: je grösser der Luftwiderstand ist, desto kleiner wird die Motorradbeschleunigung, je grösser die Motorradbeschleunigung ist, desto grösser wird die relative Geschwindigkeit und je grösser die relative Geschwindigkeit ist, desto grösser wird der Luftwiderstand. Der zweite Feedback Loop ergibt sich beim Beschleunigungswiderstand F_A: denn je grösser dieser ist, desto kleiner wird die Beschleunigung des Motorrades und je grösser diese Beschleunigung ist, desto grösser wird der Beschleunigungswiderstand.

Nun folgt der nächste Schritt der Modellbildung: die Überführung des Causal Loop Diagramms in ein Stock und Flow Diagramm (vgl. Abschnitt 2.3.1).

Abbildung 2.9 zeigt das entwickelte Stock und Flow Diagramm. Der Luftwiderstand F_L,

der Radwiderstand F_R, der Steigungswiderstand F_{St} und der Beschleunigungswiderstand F_A werden als Auxiliaries modelliert (vgl. Abschnitt 2.3.1). Die Auxiliaries werden jeweils aus verschiedenen Inputvariablen bzw. systemeigenen Levels gebildet. Der Steigungswinkel α gilt als veränderliche Rate des Levels *Steigung*, der wiederum seinen Wert über die Zeit akkumuliert (vgl. Abschnitt 2.3.1). Die Motorradbeschleunigung a_M und die Windbeschleunigung a_W werden als eigenständige Raten der Relativgeschwindigkeit v_{rel} betrachtet, da sich die Geschwindigkeit anhand von sich ändernden Beschleunigungen über die Zeit bildet. Schliesslich wird die Antriebskraft F_T (bzw. Traktionskraft am Rad) als Level der Rate der sich ändernden Motorkraft angesehen.

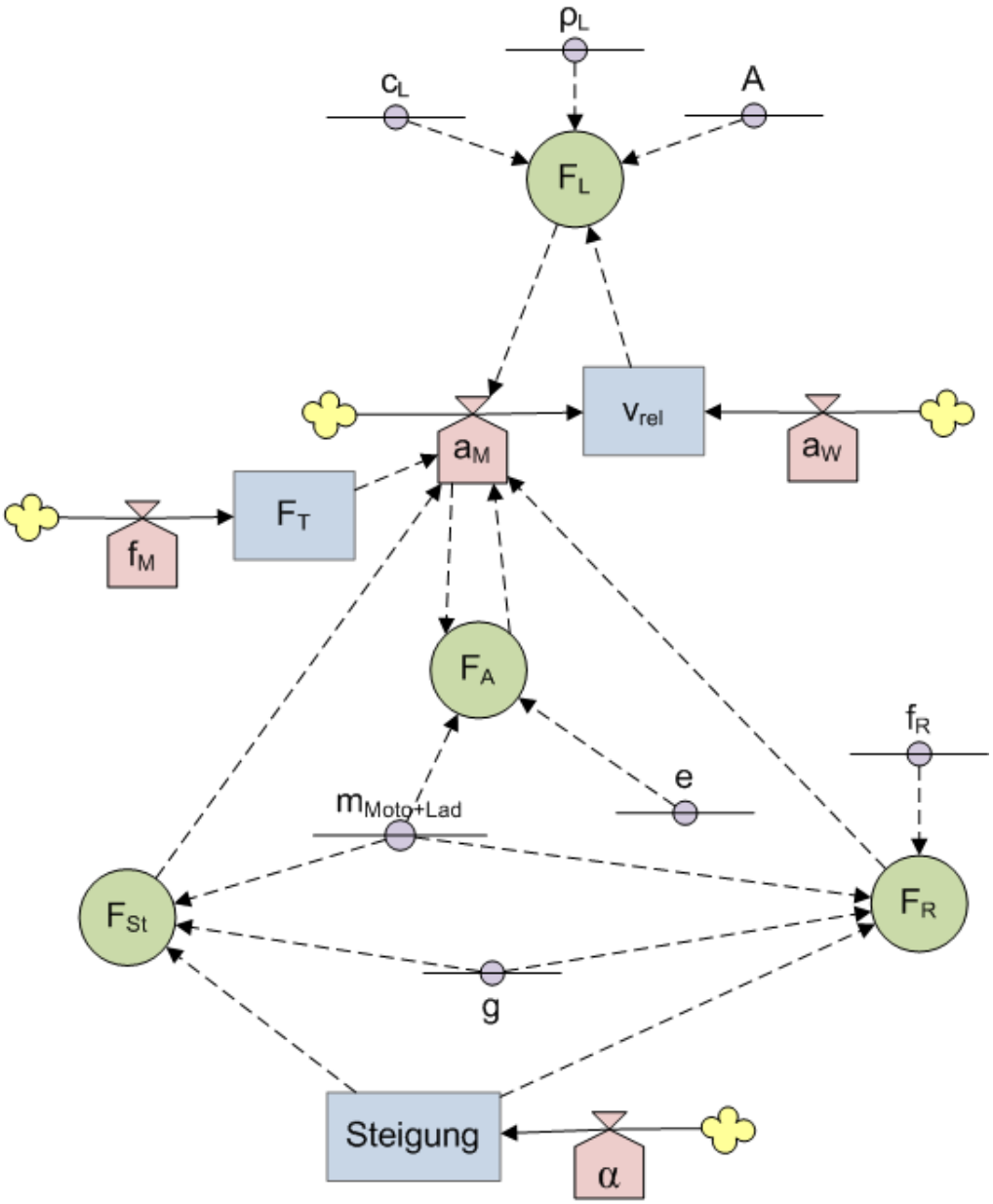


Abbildung 2.9: Stock und Flow Diagramm Motorrad

Mit Hilfe der beiden entwickelten Diagramme (Causal Loop und Stock und Flow) kann nun die in Kapitel 3 beschriebene Implementierung vorgenommen werden.

Kapitel 3

Implementierung

In diesem Kapitel wird die Implementierung des in Kapitel 2 gezeigten Simulationsmodells beschrieben. Zuerst wird eine Einführung in die verwendete Android Plattform gegeben, gefolgt von der Beschreibung der Entwicklungsumgebung und deren Tools. Danach wird das allgemeine Konzept erläutert. Dann werden Angaben zu den Input Daten gemacht und am Schluss folgen die Erklärungen zur Implementierung der Android Applikation.

3.1 Android Plattform

Die Applikation wurde für die Android Plattform entwickelt, da die Applikation auf mobilen Geräten zum Einsatz kommen soll. Android ist ein von Google herausgegebenes freies und quelloffenes Betriebssystem für mobile Geräte wie Telefone, Tablets und Netbooks, aber auch für Geräte aus den Bereichen Auto-Infotainment und Home Entertainment (Becker und Pant, 2010; Google Inc., 2011e).

Abbildung 3.1 zeigt die Architektur von Android. Der Linux-Kernel bildet als Schnittstelle zur Hardware die Basis der Android Architektur (Becker und Pant, 2010). Die Laufzeitumgebung basiert auf der Dalvik Virtual Machine (DVM) und enthält die Hauptbibliotheken (Becker und Pant, 2010). Damit die verschiedenen Applikationen aus sicherheitstechnischen Gründen nicht einen gemeinsamen Speicherbereich teilen und beim Absturz einer Applikation nicht auch andere Prozesse in Mitleidenschaft gezogen werden, läuft jede Applikation in einer eigenen DVM (Becker und Pant, 2010). Die Schicht der Standardbibliotheken beinhaltet alle C/C++ Bibliotheken, die von verschiedenen Komponenten des Android Systems benutzt werden (Google Inc., 2011g). Der Anwendungsrahmen (Application Framework) ist in Java implementiert und steht der Entwicklungsumgebung als Zugriff auf die Hardwarekomponenten zur Verfügung (Becker und Pant, 2010). Schliesslich bildet die Anwendungsschicht diejenige Schicht ab, in der sich die Android Applikationen befinden (Becker und Pant, 2010).

Android bietet ein auf Komponenten basiertes Framework für die Entwicklung an (Becker und Pant, 2010). Das Framework bietet dazu vier Komponenten an: die *Activity*, der *Service*, der *Content Provider* und der *Broadcast Receiver* (Becker und Pant, 2010). Die

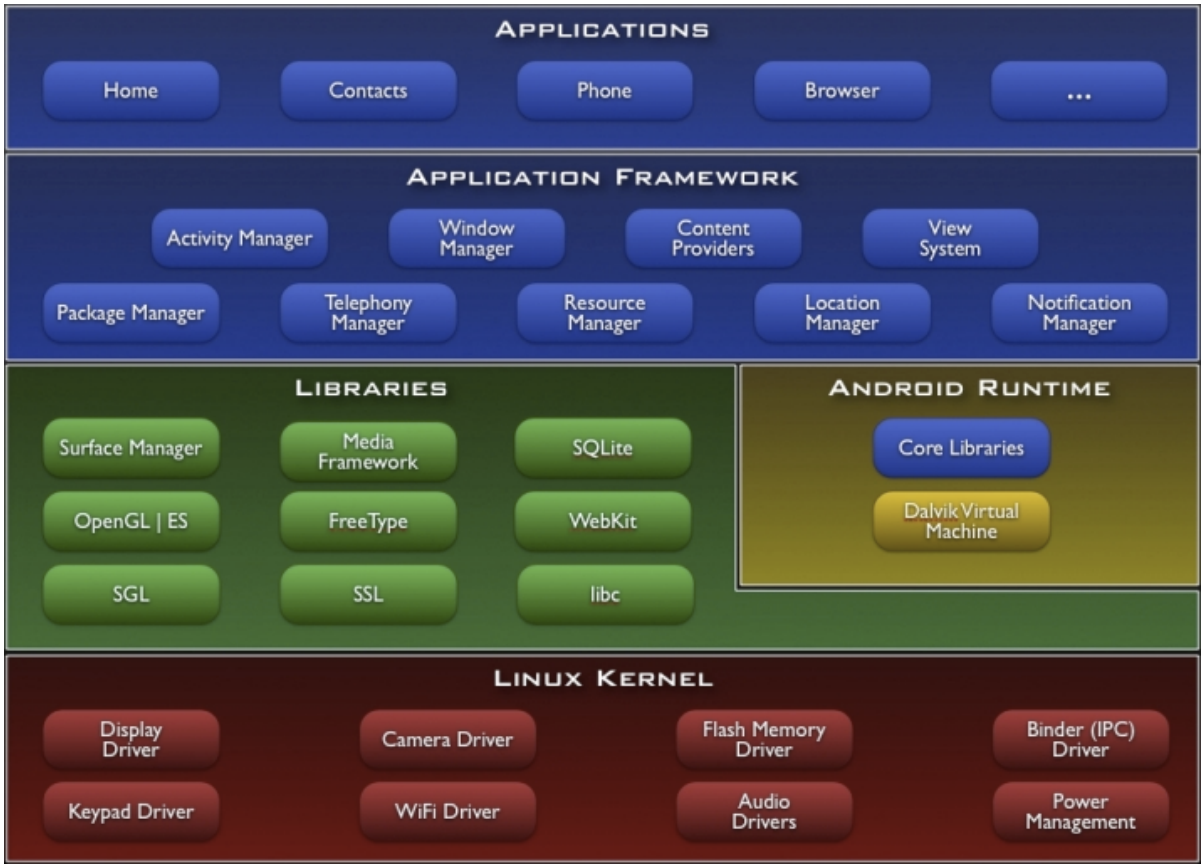


Abbildung 3.1: Android Architektur (Google Inc., 2011g)

Activity ist die wohl wichtigste Komponente und dient zur Darstellung und Verwaltung von Oberflächen (Becker und Pant, 2010). In ihr passieren die Interaktionen mit dem/-der BenutzerIn (z.B. eine SMS schreiben) (Becker und Pant, 2010). *Services* werden für Prozesse, die im Hintergrund laufen, benötigt (z.B. Musik abspielen, währendem man im Internet surft) (Becker und Pant, 2010). Ein *Content Provider* wird dann eingesetzt, wenn verschiedene Applikationen miteinander kommunizieren bzw. Daten ausgetauscht werden möchten (Becker und Pant, 2010). *Broadcast Receivers* sind – wie es der Name bereits vermuten lässt – dafür zuständig, vom Betriebssystem verteilte Nachrichten zu erhalten (z.B. wenn ein SMS eingegangen ist, schickt das Betriebssystem eine Nachricht und alle Applikationen, die dazu einen Broadcast Receiver eingerichtet haben, werden darüber informiert) (Becker und Pant, 2010).

Damit die Android Laufzeitumgebung weiss, aus welchen Komponente sich eine Applikation zusammensetzt, wird eine Datei im XML Format namens *AndroidManifest.xml* benötigt (Becker und Pant, 2010). Das Manifest enthält folgende Angaben:

- Java Paket Name
- Versionsnummer der Applikation
- Version des Android Software Development Kits (SDK)

- Deklaration der Komponenten (Activities, Services, Content Providers und Broadcast Receivers)
- Berechtigungen, die die Applikation erhält
- Zusätzliche Bibliotheken

Android Applikationen werden üblicherweise in der Programmiersprache Java geschrieben (Becker und Pant, 2010).[6] Oberflächen (also GUI Elemente) und das eben erwähnte Android Manifest werden mit Hilfe von XML Dateien deklariert (Becker und Pant, 2010). Als Datenbankbackend kommt SQLite[7] zum Einsatz (Becker und Pant, 2010). SQLite ist ein quelloffenes und ein für mobile Plattformen optimiertes relationales Datenbanksystem (Becker und Pant, 2010).

Google erhebt periodisch statistische Daten darüber, welche Android Version am häufigsten verwendet wird (Google Inc., 2011f). Gemäss der Erhebung per 3. Oktober 2011 (über einen zweiwöchigen Zeitraum) ist die Version 2.2 mit 45.3% das meist benutzte Betriebssystem innerhalb der Android Plattformfamilie, gefolgt von der Version 2.3.3 mit 38.2% und der Version 2.1 mit 11.7% (Google Inc., 2011f).

3.2 Entwicklungsumgebung

Als Entwicklungsumgebung wurde die von Google empfohlene Umgebung Eclipse[8] gewählt (Google Inc., 2011d). Mit dem Android SDK liefert Google sowohl die Application Programming Interfaces (API) der verschiedenen Android Versionen, als auch den Standalone Android Emulator aus (Google Inc., 2011d). Abbildung 3.2 zeigt den Emulator, der es ermöglicht, direkt auf dem Computer ein Android Telefon und dessen Applikationen zu emulieren (Becker und Pant, 2010). Es wurde zudem das von Google entwickelte *ADT Plugin* für Eclipse verwendet, das die Entwicklung von Android Applikationen erleichtert und ermöglicht, den Android Emulator direkt aus Eclipse heraus zu starten, zu verwalten und zu debuggen (Google Inc., 2011a,d). Zusätzlich kann auch per Kommandozeile auf den laufenden Emulator zugegriffen werden (Becker und Pant, 2010).

3.3 Konzept

Dieser Abschnitt beschreibt das Konzept, das zur Implementierung des in Kapitel 2 entwickelten Modells verwendet wird.

Wie bereits in der Einleitung im Kapitel 1 erwähnt wurde, hat die Empa eine Sensorenbox entwickelt, die es ermöglicht, während der Fahrt mit einem Elektromotorrad Daten zu

[6] Google stellt zusätzlich ein Native Development Kit (NDK) zur Verfügung, mit dem man in C oder C++ nativen Code den Applikationen hinzufügen kann (Google Inc., 2011h).

[7] Mehr zu SQLite unter `http://www.sqlite.org`.

[8] Mehr zu Eclipse unter `http://www.eclipse.org`.

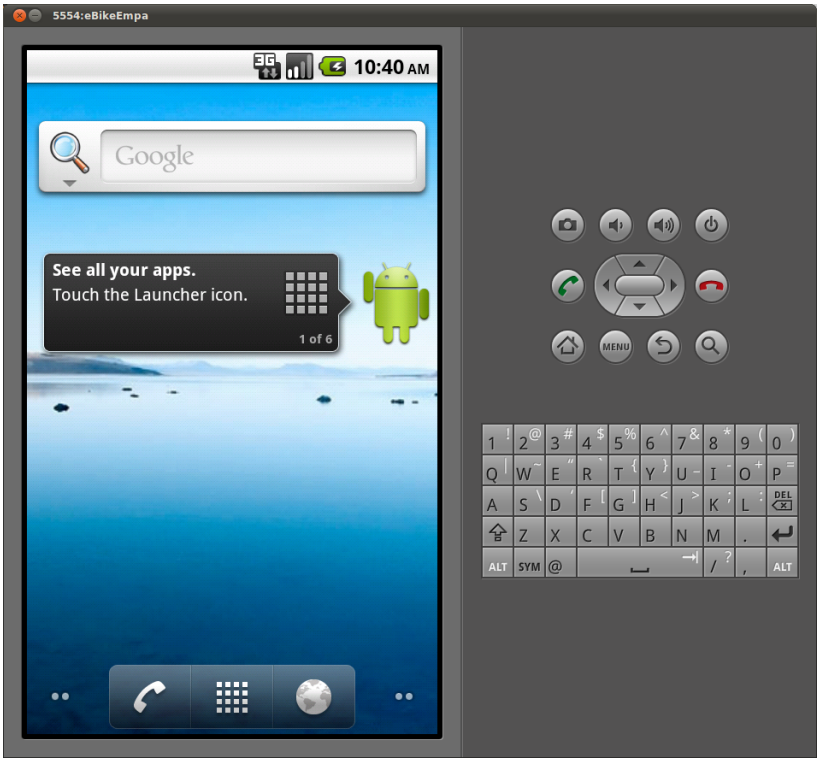


Abbildung 3.2: In Ubuntu ausgeführter Android Emulator

messen und zu sammeln. Ziel der nachfolgend beschriebenen Implementierung ist es, die Messdaten in einer Android Applikation verwendbar zu machen. Dazu sollen die Messdaten in die Applikation eingelesen werden und dem Benutzer/der Benutzerin der Android Applikation eine entsprechende Oberfläche zur Verfügung gestellt werden, in der die nachfolgend festgelegten Anwendungsfälle ausgeführt werden können.

Abbildung 3.3 zeigt alle Anwendungsfälle, die implementiert werden. Der Benutzer erstellt ein Projekt und legt Parameter für die Berechnung der in Abschnitt 2.2 erwähnten Kräfte fest. Die gesammelten Sensorendaten werden dann in das Projekt eingelesen. Der Benutzer kann jederzeit bereits erstellte Projekte und deren Parameter bearbeiten. Projekte können auch entfernt werden, wobei gleichzeitig auch bereits erstellte Simulationen, die sich auf das Projekt beziehen, entfernt werden. Der Benutzer/die Benutzerin kann Simulationen, die die Fahrt eines Projektes mit anderen Parametern simulieren, erstellen, bearbeiten oder entfernen. Die eingelesene Fahrt kann sowohl in Google Maps betrachtet werden als auch in einem zweidimensionalen Koordinatensystem mit den Simulationen verglichen werden. Das gebildete Koordinatensystem kann als Grafikdatei exportiert werden.

Es soll hier festgelegt werden, was ein Projekt repräsentiert. Ein Projekt basiert auf den Inputdaten einer realen Fahrt (die Messdaten dieser Fahrt wurden mit der Sensorenbox gesammelt). Eine reale Fahrt kann mit anderen Parametern simuliert werden. Das Projekt enthält deshalb zusätzlich zur realen Fahrt alle Simulationen, die auf eben dieser Fahrt basieren.

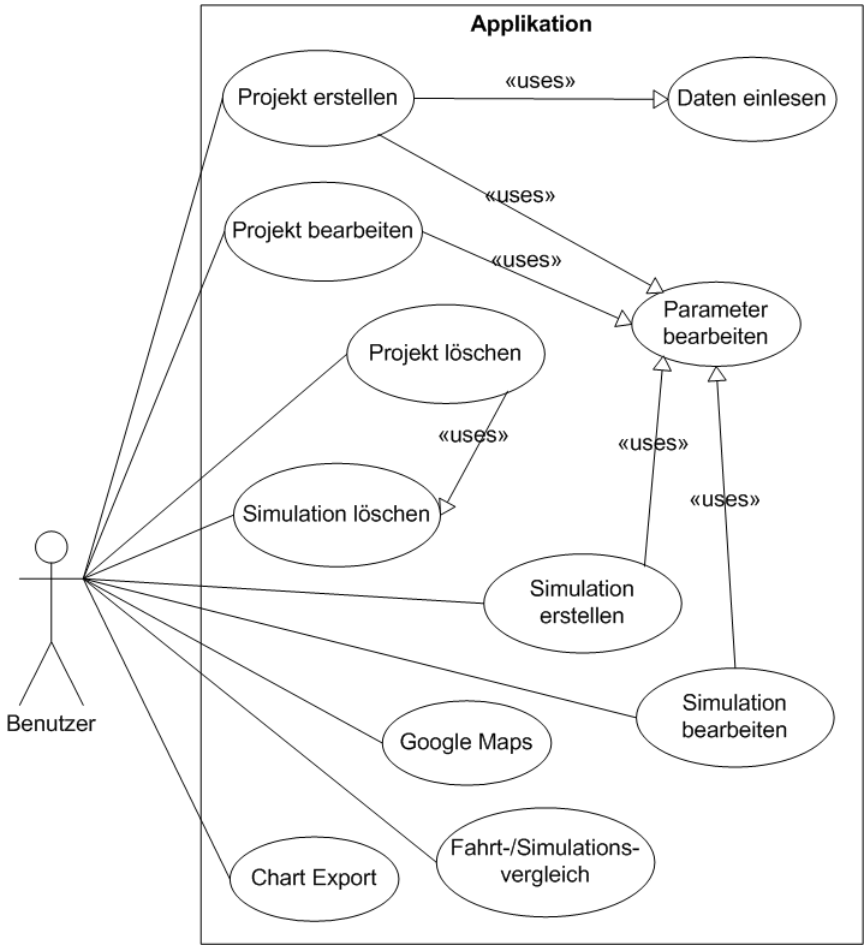


Abbildung 3.3: Anwendungsfälle der Android Applikation

3.4 Inputdaten

Die Applikation ist auf Inputdaten angewiesen (siehe Abschnitt 3.3). Die Gewinnung der Inputdaten ist nicht Gegenstand dieser Implementierung. Es wird davon ausgegangen, dass die gesammelten Daten aus der Sensorenbox extrahiert wurden und der Applikation im CSV-Format zur Verfügung stehen. Deshalb werden in diesem Abschnitt lediglich diejenigen Werte beschrieben, die der Inputdaten-Datei mindestens enthalten sein müssen. Folgende Werte müssen zur Verfügung stehen[9]:

- **Zeitstempel**, der die Zeit der Fahrt seit Messbeginn kumuliert, in Sekunden
- **Zeit nach Mitternacht** in Sekunden
- **Relativgeschwindigkeit** des Motorrades in km/h
- **Höhe über Meer** in Metern

[9]Eigentlich müssten für die aktuelle Implementierung weniger Werte zwingend vorhanden sein (siehe Abschnitt 3.5). Um spätere Implementierungsanpassungen zu vereinfachen, sollen aber bereits bei der Initialimplementierung die wichtigsten Werte der Sensorenbox verarbeitet werden.

- **GPS Längengrad** in Dezimalgrad
- **GPS Breitengrad** in Dezimalgrad
- **A00Voltage**
- **A01Voltage**
- **VBat**

Die drei letztgenannten Werte (A00Voltage, A01Voltage und VBat) beschreiben Sensorenwerte, die sich auf die Batterie des Elektroantriebes beziehen. Alle aufgelisteten Werte müssen als Dezimalzahlen vorhanden sein.

Da es gemäss den Anforderungen der Empa nicht gewährleistet ist, dass die Spaltenwerte der Inputdatendatei jeweils identisch sortiert bzw. vorhanden sind, kann die Datendatei neben den zwingend erforderlichen Werten zusätzliche Werte enthalten. In Abschnitt 3.5 wird das Mapping der Datendatei beschrieben, welches ermöglicht, pro Projekt die Spaltenanordnung der Datendatei individuell festzulegen.

Eine für diese Arbeit von der Empa zur Verfügung gestellte Inputdatendatei befindet sich auszugsweise im Anhang A dieser Arbeit.

3.5 Applikation

Dieser Abschnitt befasst sich mit der eigentlichen Implementierung der Applikation. Zuerst werden allgemeine Hinweise dazu gegeben. Danach werden der Aufbau und die Funktionalitäten der Applikation beschrieben. Schliesslich werden das Datenmodell und die Datenbankimplementierung erläutert.

3.5.1 Allgemeine Hinweise

Die hier beschriebene Applikation wurde auf Basis der SDK-Version 2.2 entwickelt und ist somit gemäss Abschnitt 3.1 mit mindestens 83.5% der aktuell genutzten Android Geräte kompatibel. Die Applikation verwendet ausserdem die Google API, die eine Schnittstelle zu Google Diensten und Bibliotheken, wie z.B. Google Maps, zur Verfügung stellt (Google Inc., 2011c).

Die Applikation besteht aus mehr als 3'000 puren Java Codezeilen, aus mehr als 600 Kommentarzeilen und aus ein paar Hundert Zeilen XML[10].

Kommentare innerhalb des Quellcodes sind in Englisch verfasst. Ebenfalls sind die auf der Oberfläche dargestellten Zeichenketten in Englisch geschrieben. Die Zeichenketten werden

[10]Zur Berechnung der Anzahl Codezeilen, d.h. ohne Kommentare und leeren Zeilen, und des Verhältnisses zu den Kommentaren wurde das Eclipse Plugin *CodePro AnalytiX* von Google verwendet. Mehr dazu unter `http://code.google.com/intl/de-DE/javadevtools/download-codepro.html`.

bei der Android Applikation vom Quellcode getrennt in einer XML-Datei festgehalten (Becker und Pant, 2010). Innerhalb des Codes wird auf die XML-Datei referenziert (Becker und Pant, 2010). Auf der Oberfläche vorkommende Zeichenketten können so relativ einfach übersetzt werden: es wird dazu die XML-Datei kopiert, nach einer vom Android Framework vorgegebenen Konvention umbenannt und übersetzt (Becker und Pant, 2010). Die Laufzeitumgebung stellt dem/der BenutzerIn je nach Spracheinstellung des jeweiligen Android Gerätes die Zeichenketten in der entsprechenden Sprache dar (Becker und Pant, 2010).

Auf einem mobilen Gerät stehen nicht die gleichen Hardwareressourcen wie auf einem Desktop- oder Serversystem zur Verfügung (Becker und Pant, 2010). Der Speicherverbrauch und die Prozessorlast muss demzufolge möglichst gering gehalten werden (Becker und Pant, 2010). Ausserdem bezieht ein mobiles Gerät (wenn es nicht gerade aufgeladen wird) Strom von „einer sich schnell erschöpfenden Stromquelle“ (Becker und Pant, 2010), vom Akku nämlich. Um sparsam mit den Ressourcen umzugehen, werden Ansätze, die typischerweise bei der objektorientierten Programmierung eingesetzt werden, entweder gar nicht oder nur spärlich eingesetzt (Becker und Pant, 2010). Auf die Verwendung von Interfaces und Getter-/Setter-Methoden soll aus Gründen der Performance und des Speicherverbrauchs möglichst verzichtet werden (Becker und Pant, 2010). Aufwändige Datenbankzugriffsschichten sollen ebenfalls vermieden werden, damit die Applikation nicht zu gross und nicht zu langsam wird (Becker und Pant, 2010). Zu guter Letzt soll die Anzahl der Objekterzeugung möglichst gering gehalten werden, da Objekte in den Speicher geladen werden und dieser, wie erwähnt, stark limitiert ist (Becker und Pant, 2010). In den folgenden Abschnitten wird auf Kompromisse dieser Art hingewiesen.

In Abschnitt 3.1 wurde gezeigt, dass das Android Framework vier Komponenten anbietet (Activity, Service, Content Provider und Broadcast Receiver). Jede Komponente hat einen Lebenszyklus. Sie durchläuft von der Erstellung bis zur Beendung – je nach Art der Komponente – verschiedene Zustände (Becker und Pant, 2010). Komponenten bestimmen ihren Lebenszyklus nicht autonom, sondern sind von systemweiten Ereignissen abhängig (Becker und Pant, 2010). Ein eingehender Telefonanruf, beispielsweise, überdeckt die gerade aktive Applikation (Becker und Pant, 2010). Oder falls gerade eine Activity angezeigt wird und das Telefon gedreht wird (und deshalb auch der Bildschirm), wird die Activity neu aufgebaut (Becker und Pant, 2010). Die Komponenten bieten dazu eigene Methoden an, damit bei solchen Ereignissen und Unterbrechungen keine ungespeicherten Daten verloren gehen (Becker und Pant, 2010). Die vorliegende Implementierung nutzt teilweise die von den Komponenten zur Verfügung gestellten Methoden ihres Lebenszyklus. Der Schwerpunkt der Implementierung lag aber nicht darin, alle möglichen Ereignisse abzufangen und korrekt verarbeiten zu können. Das heisst, dass bei der Benutzung der vorliegenden Applikation bei bestimmten Ereignissen die Applikation „fehlerhaftes“ Verhalten aufweist und sie deshalb bei einer späteren Überarbeitung dahingehend noch verbessert werden sollte.

3.5.2 Aufbau

Thematisch zusammengehörende Klassen werden bei der Java Programmierung in Paketen gruppiert (Ullenboom, 2008). In der Android Programmierung verhält es sich nicht

anders (Becker und Pant, 2010). Die Tabelle 3.1 zeigt die verwendeten Pakete und eine kurze Beschreibung derselben.

Paket	**Beschreibung**
ch.marcocreola.android.**ebike**	Beinhaltet die wichtigsten Programm Activities, die es z.B. für die Projekterstellung, -editierung und -anzeige benötigt
ch.marcocreola.android.**ebike.charts**	Enthält die Chart-Funktionalitäten
ch.marcocreola.android.**ebike.database**	Vereinigt Funktionalitäten zur Erstellung der Datenbank und deren Zugriff
ch.marcocreola.android.**ebike.gps**	Enthält Berechnungsmethoden für GPS Werte, aber auch die Google Maps Darstellung einer Fahrt
ch.marcocreola.android.**ebike.helper**	Beinhaltet verschiedene Hilfsmethoden und -activities
ch.marcocreola.android.**ebike.physics**	Enthält die Funktionalitäten zur Berechnung der in Abschnitt 2.2 beschriebenen Kräfte

Tabelle 3.1: Java Pakete und deren Dateien

Wie in Abschnitt 3.1 erwähnt, kommen bei der Android Programmierung XML Dateien für die Oberflächengestaltung hinzu. Diese werden hier nicht einzeln aufgelistet, da es sich um einfache XML-Deklarationen handelt.

In Android erhält jede Applikation eine eigene Datenbank (Becker und Pant, 2010). So wird sichergestellt, dass nur die zu den Daten gehörende Applikation Zugriff auf dieselben erhält (Becker und Pant, 2010). Gemäss der offiziellen Developer Dokumentation von Google (2011b) wird SQLite in der Version 3.4.0 ausgeliefert. Tatsächlich ist es aber so, dass je nach Android SDK-Version unterschiedliche SQLite Versionen zum Einsatz kommen[11]. Tabelle 3.2 zeigt die drei meist verwendeten Android Plattform (siehe Abschnitt 3.1) und deren SQLite Version. In Abschnitt 3.5.1 wurde festgehalten, dass die Implementierung die zwei meist verwendeten Android Plattformen unterstützen sollte. Deshalb baut die Datenbankimplementierung auf der SQLite Version 3.6.22 auf.

Android SDK-Version	**SQLite Version**
2.1	3.5.9
2.2	3.6.22
2.3.3	3.6.22

Tabelle 3.2: Android SDK-Version und SQLite Version

[11]Dazu gibt es von Google keine offizielle Bestätigung. Wenn man aber mit der Kommandozeile auf die gerade emulierte Android Umgebung zugreift, kann man die Datenbankversion abfragen. Macht man dies mit unterschiedlichen SDK-Versionen, wird man feststellen, dass unterschiedliche SQLite Versionen eingesetzt werden. Dieser Umstand wird auf Webforen und -blogs ebenfalls bestätigt (Simpligility Technologies Inc., 2010; Stack Exchange Inc., 2011).

3.5.3 Funktionalitäten

Massgebend für die Funktionalitäten sind die in Abschnitt 3.3 aufgezählten Anwendungsfälle. Die folgende Liste zählt die Anwendungsfälle nochmals auf. In Klammern stehen dabei die Nummern derjenigen Abschnitte, in denen der Anwendungsfall beschrieben wird.

- Projekt erstellen, bearbeiten und entfernen (3.5.3.1)
- Daten einlesen (3.5.3.1)
- Parameter bearbeiten (3.5.3.1 und 3.5.3.2)
- Simulation erstellen, bearbeiten und entfernen (3.5.3.2)
- Google Maps (3.5.3.3)
- Fahrt- und Simulationsvergleich (3.5.3.4)
- Chart Export (3.5.3.4)

3.5.3.1 Projekt erstellen, bearbeiten und entfernen

Startet man die Applikation, wird die Activity *ProjectsListActivity* im Paket *ebike*[12] angezeigt. Abbildung 3.4a zeigt diese Activity als leere Liste an, da noch keine Projekte angelegt wurden. Um nun ein solches Projekt zu erstellen, wählt man im *Menu* (via Menu Taste) den Eintrag *Add Project* aus (siehe Abbildung 3.4b).

Die Activity, die sich dazu öffnet (*ProjectCreateActivity* im Paket *ebike*), wird in Abbildung 3.5a dargestellt[13]. Bei der Projekterstellung wird der Projektname, die einzulesende Datei mit den Inputdaten, die erste Datenzeile der Inputdatei, das Spaltenmapping, das Voltagemapping und die Parameter festgelegt.

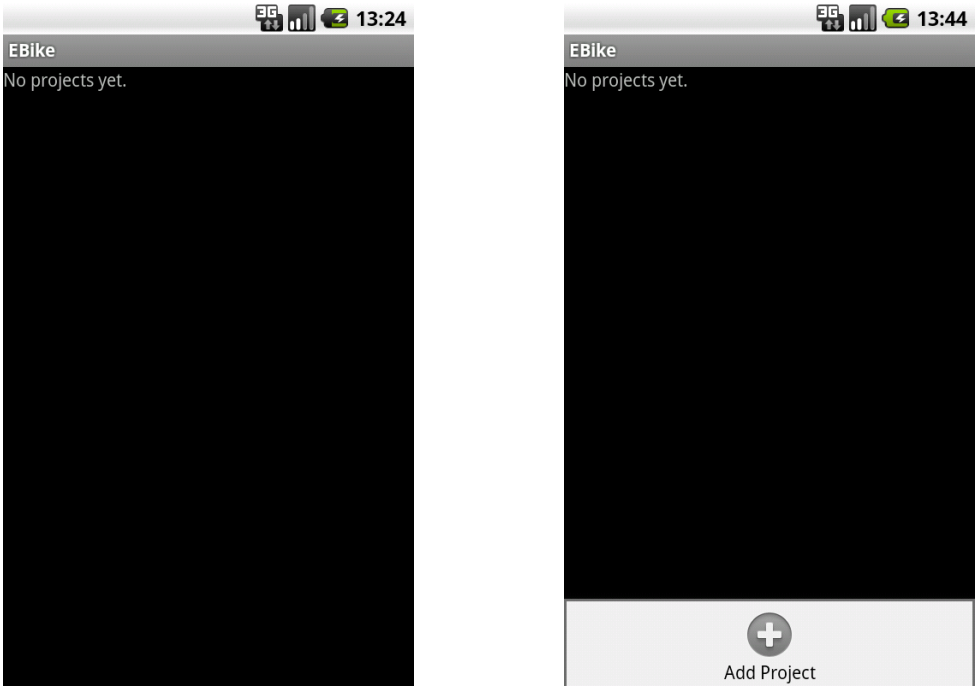
Unter *Choose File* öffnet sich die Activity *FileBrowserActivity*[14] (Paket *ebike.helper*), in der das Wurzelverzeichnis der gemounteten SD-Karte[15] angezeigt wird (siehe Abbildung 3.5b). Die SD-Karte kann im Emulator ebenfalls emuliert werden und es lassen sich direkt aus Eclipse heraus Dateien von und zur SD-Karte kopieren (Becker und Pant, 2010). Eine

[12] Die Paketnamen werden hier und nachfolgend zur besseren Lesbarkeit abgekürzt, d.h. die Paketnamen repräsentieren jeweils Unterpakete von *ch.marcocreola.android*.

[13] Das „+/-“ Layoutelement, das auf dieser Oberfläche und auf anderen verwendet wird, wurde mit Hilfe der Open Source FloatPickerWidget API erstellt. Die API wurde dem Eclipse Projekt als externe Bibliothek hinzugefügt. Mehr zur dieser API unter `http://www.laurencegellert.com/software/android-float-picker-widget/`.

[14] Die Klasse *FileBrowserActivity* wurde aus einem Developer Blog hinsichtlich eines Android Tutorials übernommen und angepasst. Mehr dazu unter `http://wordpress.the-engine.at/2011/05/ein-einfacher-android-filebrowser`.

[15] Die meisten Android Geräte sind ab Werk mit einem externen Speicher in Form einer SD-Karte ausgerüstet (Becker und Pant, 2010). Das eben kürzlich vorgestellte Galaxy Nexus Telefon von Samsung, das mit der neuesten Android Version 4.0 erscheint, besitzt beispielsweise keinen Slot für SD-Karten (heise online, 2011).



(a) Leere Projektliste (b) Menu Leiste auf der Startseite

Abbildung 3.4: Startseite mit Projektliste und Menu Leiste

zuvor auf die SD-Karte kopierte Inputdatendatei kann so für die Projekterstellung ausgewählt werden. Ein Projekt kann nicht erstellt werden, falls keine Datei ausgewählt wurde – eine entsprechende Fehlermeldung wird in diesem Fall dem/der BenutzerIn angezeigt.

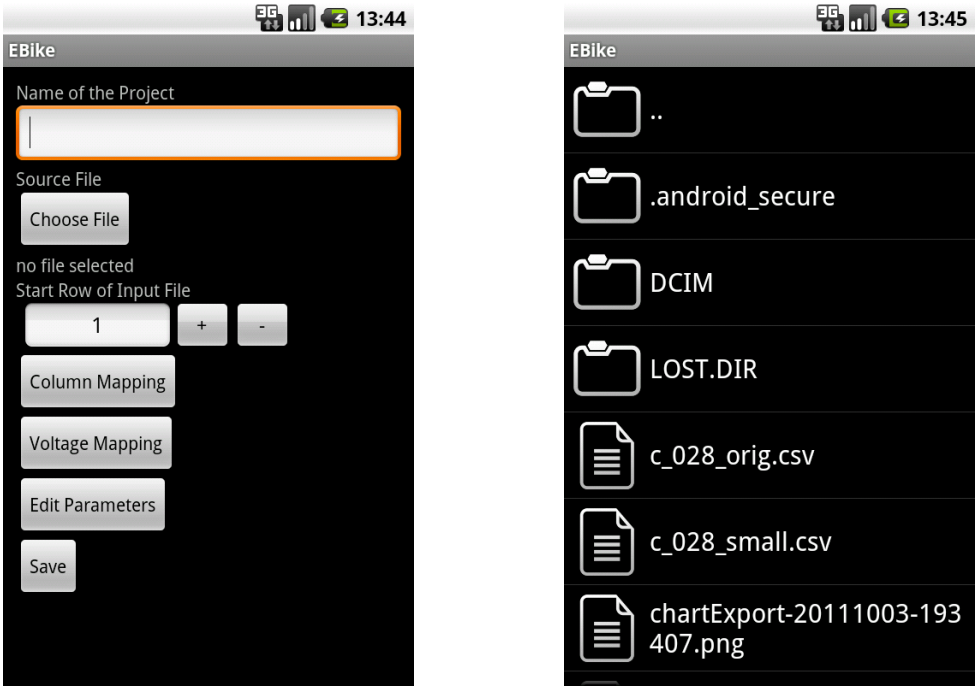
Unter *Start Row of Input File* wird die Zeilennummer der gerade gewählten Datei angegeben, ab der die tatsächlichen Inputdaten beginnen; d.h. allfällige Spaltenköpfe werden so übersprungen und nicht verarbeitet.

Wie in Abschnitt 3.4 beschrieben, bezieht sich eine der Implementierungsanforderungen auf individuell wählbare Spalten der jeweiligen Inputdatendatei. Das kann in der Applikation mit Hilfe des *Column Mappings* festgelegt werden. Abbildung 3.6a zeigt den Dialog, in dem nun alle als zwingend festgelegten Spalten (siehe Abschnitt 3.4) ausgewählt werden können. Falls eine Spalte für mehrere Werte definiert wird (z.B. den Spalten *Time* und *GPS Longitude* würde die gleiche Spalte vergeben werden), wird dem/der BenutzerIn eine entsprechende Fehlermeldung angezeigt.

Im Dialog *Voltage Mapping* (siehe Abbildung 3.6b) wird das in Abschnitt 3.4 angesprochene Mapping der Sensorenwerte der Batterie angegeben.

Unter *Edit Parameters* (siehe Abbildung 3.7a) werden folgende Werte festgelegt: Gewicht des Motorrades und des Fahrers/der Fahrerin, der Luft- und Rollwiderstandskoeffizient, die Stirnfläche und den Massenträgheitsfaktor (vgl. dazu Abschnitt 2.2).

Bei der Speicherung des Projektes wird die Datendatei eingelesen und verarbeitet. Abbildung 3.7b zeigt den Informationsdialog, der beim Speichern des Projektes erscheint.



(a) Projekt erstellen (b) Der Filebrowser

Abbildung 3.5: Projekterstellung und Auswahl der Inputdatendatei

(a) Spaltenmapping (b) Voltagemapping

Abbildung 3.6: Die beiden Dialoge zum Mapping bei der Projekterstellung

Der Speichernvorgang wird in der Methode *run* der Klasse *ProjectCreateActivity* als nebenläufigen Thread im gleichen Prozess ausgeführt. Zuerst werden der Projektname, der Pfad zur Inputdatendatei, die Startzeile und die Voltagemapping-Werte gespeichert. Da-

nach werden das Spaltenmapping und die Parameterwerte in die Datenbank eingefügt. Die Datei wird mit der Methode *parseInputFile*, die aus der *run*-Methode gestartet wird, zeilenweise eingelesen. Pro Zeile werden die in Abschnitt 3.4 benötigten Inputwerte anhand des Spaltenmappings zugeordnet. Zu guter Letzt werden die physikalischen Kräfte berechnet und in die Datenbank geschrieben. Abschnitt 3.5.3.5 beschäftigt sich mit der Berechnung der physikalischen Kräfte, die beim Speichern des Projektes zur Anwendung kommt.

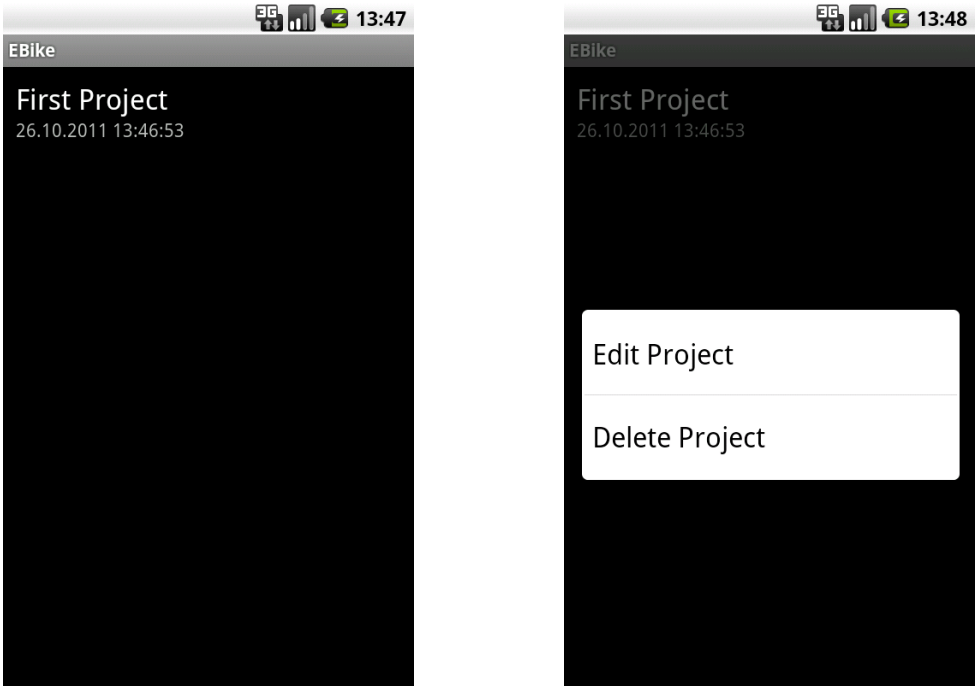
(a) Bearbeitung der Parameter (b) Speichernvorgang

Abbildung 3.7: Parameterbearbeitung und der Speicherndialog

Nachdem ein Projekt erstellt wurde, wird dieses auf der Startseite (*ProjectsListActivity*) mit dem Namen und des Datums der letzten Änderung angezeigt (siehe Abbildung 3.8a). Wenn auf ein Projekteintrag lang gedrückt wird, erscheint das Kontextmenu (siehe Abbildung 3.8b), in dem das Projekt entweder bearbeitet oder entfernt werden kann.

Abbildung 3.9a zeigt die Oberfläche der Activity *ProjectEditActivity* aus dem Paket *ebike*, die sich bei der Projektbearbeitung öffnet. Die Klasse *ProjectEditActivity* ist eine Unterklasse von *AbstractEditActivity*. Die Projektbearbeitung baut auf einem ähnlichem Konstrukt wie die Simulationserstellung und -bearbeitung (siehe Abschnitt 3.5.3.2). Deshalb wurde hier eine Oberklasse mit entsprechenden Unterklassen gebildet.

Bei einem Projekt können nicht alle – zuvor bei der Erstellung gewählten – Einstellungen bearbeitet werden. Es können nur der Projektname und die Parameterwerte geändert werden. Beispielsweise würde die Änderung des Spaltenmappings ein erneutes Einlesen der Inputdatendatei benötigen. Da nach der Projekterstellung nicht sichergestellt werden kann, dass die Datei noch vorliegt, wird deshalb auf diese Funktion verzichtet. Analog dazu verhält es sich mit dem Voltagemapping: eine Änderung dazu würde nur Sinn machen,



(a) Projektliste mit einem Projekt (b) Kontextmenu eines Projektes

Abbildung 3.8: Projektliste und das Projekt Kontextmenu

falls die Inputdaten abermals eingelesen werden könnten. Zusätzlich würde der Bearbeitungsprozess erschwert werden, da allfällige Simulationen, die auf den Originaldaten der Inputdatei beruhen, neu berechnet werden müssten (vgl. Abschnitt 3.5.3.2). Wird ein Projekt nach der Bearbeitung gespeichert, werden sämtliche physikalischen Kräfte anhand der geänderten Parameter neu berechnet.

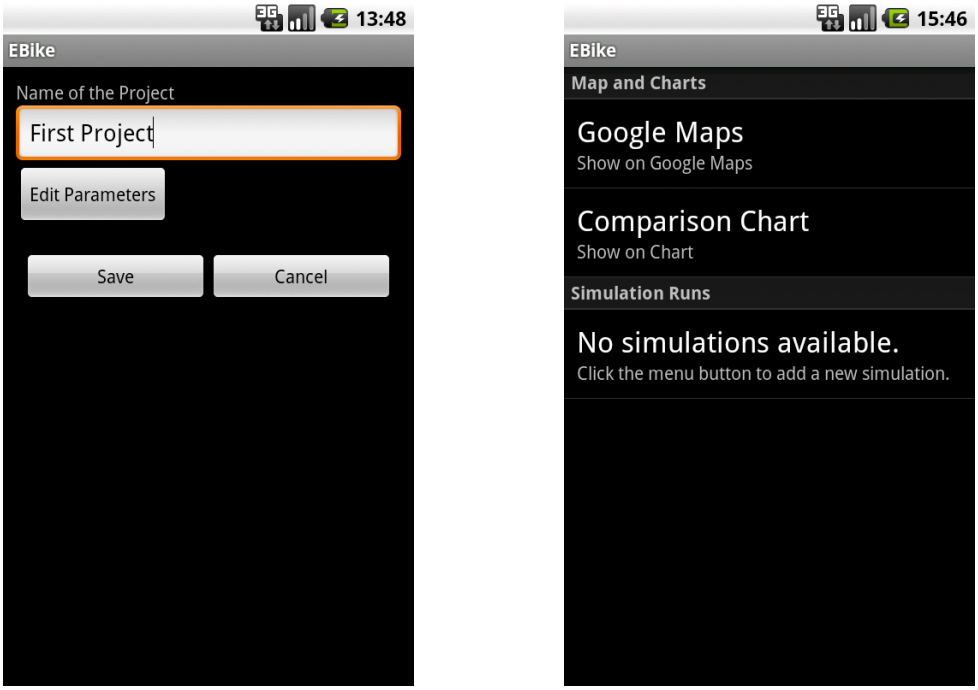
Soll ein Projekt entfernt werden, ruft man das Kontextmenu (siehe Abbildung 3.8b) auf. Es werden sämtliche Daten, die zum Projekt gehören, entfernt. Allfällig erstellte Simulationen werden dabei ebenfalls entfernt.

Wird ein Projekt in der Projektliste ausgewählt, wird die Activity *ProjectShowActivity* im Paket *ebike* aufgerufen. Dabei öffnet sich die Oberfläche der Projektübersicht (siehe Abbildung 3.9b)[16]. Im Menu dieser Activity (siehe Abbildung 3.10a) können die Aktionen *Edit Project* und *Delete Project* ausgeführt werden, die die gleichen Funktionen haben, wie die gerade erwähnten bezüglich der Projektbearbeitung und -entfernung.

3.5.3.2 Simulation erstellen, bearbeiten und entfernen

Die Applikation bietet die Möglichkeit, eine reale Fahrt, von der ja bei der Projekterstellung Messdaten eingelesen wurden, mit anderen Parametern zu simulieren. Eine Simula-

[16]Für die Darstellung der Oberfläche der Projektübersicht wurde Code eines Android Tutorials verwendet. Mehr zum Tutorial unter `http://jsharkey.org/blog/2008/08/18/separating-lists-with-headers-in-android-09/`.



(a) Projektbearbeitung (b) Projektübersicht

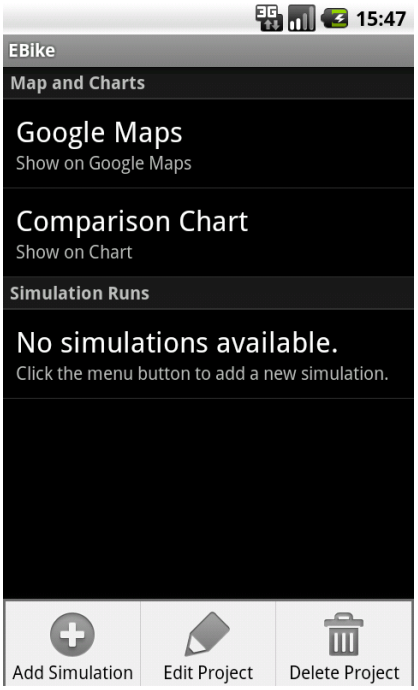
Abbildung 3.9: Projektbearbeitung und -übersicht

tion gilt jeweils für das ihr zugrundeliegende Projekt. Es gibt keine Beschränkung in der Anzahl der Simulationen.

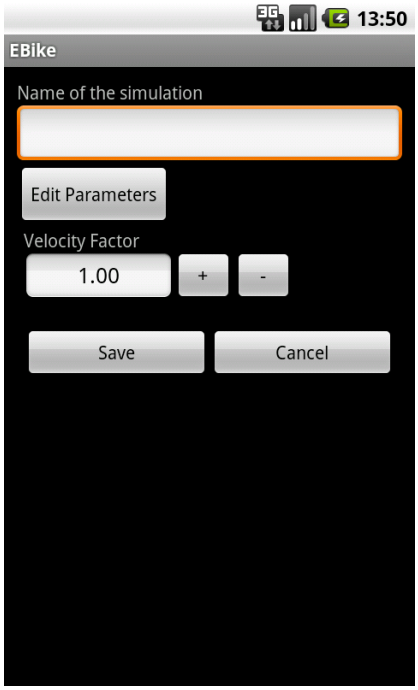
Eine Simulation fügt man dem Projekt hinzu, indem im Menu der Projektübersicht *Add Simulation* ausgewählt wird (siehe Abbildung 3.10a). Daraufhin öffnet sich die Activity *SimulationCreateRunActivity* des Paketes *ebike*, die von der Oberklasse *AbstractEditActivity* erbt. Zu einer Simulation gehören ein Name, verschiedene Parameter und der Geschwindigkeitsfaktor. Mit dem Geschwindigkeitsfaktor wird festgelegt, wie sich die Relativgeschwindigkeit des Motorrades gegenüber der tatsächlichen Fahrt ändern soll. Bei den Parametern kann man – analog zur Projekterstellung – das Gewicht des Motorrades und des Fahrers/der Fahrerin, den Luft- und Rollwiderstandskoeffizient, die Stirnfläche und den Massenträgheitsfaktor festlegen. So lässt sich eine reale Fahrt unter anderen Bedingungen simulieren. Wird die Simulation gespeichert, werden Teile der in Abschnitt 3.5.3.5 beschriebenen Vorgänge ausgeführt.

Nach dem Speichern gelangt man wieder auf die Projektübersicht, in der nun die gespeicherte Simulation angezeigt wird (siehe Abbildung 3.11a). Die Simulation kann auf zwei Arten bearbeitet werden: a) indem sie in der Projektübersicht ausgewählt wird oder b) indem das Kontextmenu der Simulation aufgerufen wird und dort *Edit Simulation* gewählt wird (siehe Abbildung 3.11b). Für die Bearbeitung wird die Activity *SimulationEditRunActivity* aufgerufen, die ebenfalls eine Unterklasse von *AbstractEditActivity* ist.

Soll eine Simulation entfernt werden, geschieht dies über das Kontextmenu und *Delete Simulation* (siehe Abbildung 3.11b).

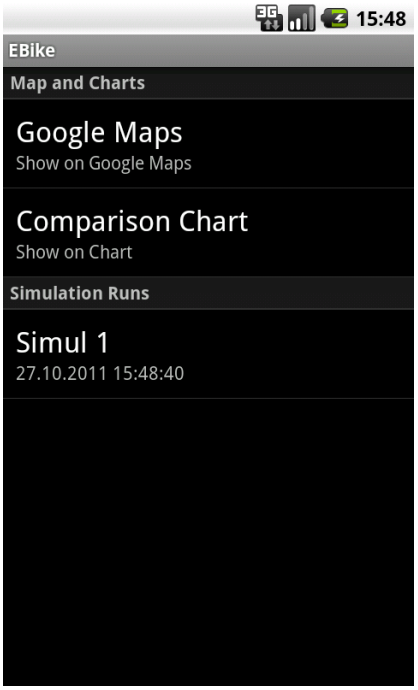


(a) Menu in der Projektübersicht

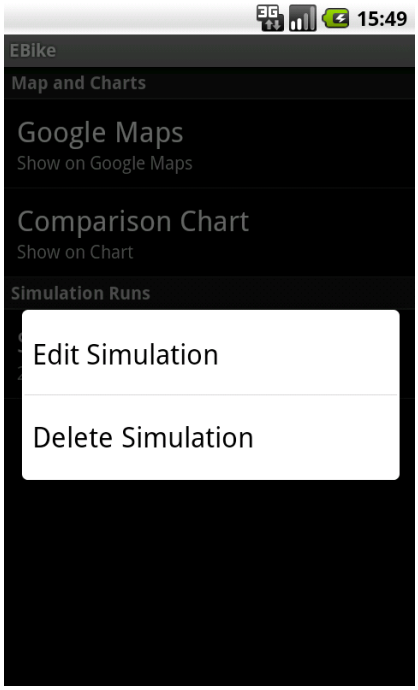


(b) Simulation erstellen/bearbeiten

Abbildung 3.10: Menu der Projektübersicht und Simulationserstellung/-bearbeitung



(a) Projektübersicht mit Simulation



(b) Kontextmenu einer Simulation

Abbildung 3.11: Projektübersicht und Kontextmenu

3.5.3.3 Anzeige in Google Maps

Wählt man in der Projektübersicht (siehe Abbildung 3.9b) *Google Maps*, kann die dem Projekt zugrundeliegende Fahrt in Google Maps betrachtet werden. Es wird die Activity

ShowMapActivity im Paket *ebike.gps* geöffnet. Dank der Google API (vgl. Abschnitt 3.5.1) steht der Android Entwicklung die Anbindung an Google Maps zur Verfügung. So wird in der *ShowMapActivity* die Klasse *OverlayLine* (paketidentisch) benutzt, um von den gespeicherten GPS Koordinaten eine Route in Google Maps zu zeichnen. In der vorliegenden Implementierung wird die Route angezeigt und der/die BenutzerIn hat die Möglichkeit, den Kartenausschnitt zu verschieben, zu vergrössern oder zu verkleinern. Abbildung 3.12 zeigt die Testfahrt auf dem Gelände der Empa in St. Gallen.

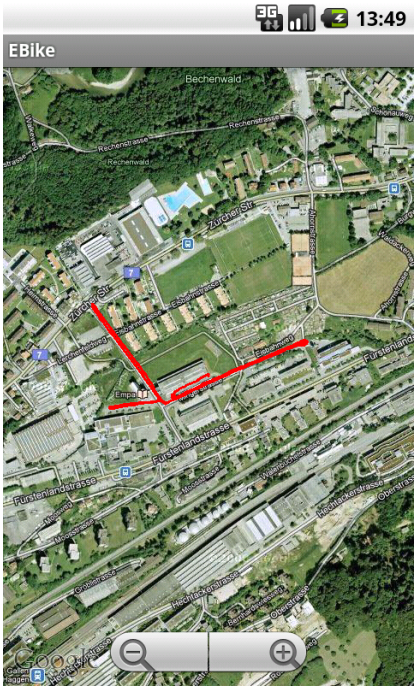


Abbildung 3.12: Fahrt in Google Maps

Auflistung 3.1 zeigt einen Auszug der Klasse *ShowMapActivity*. Die beiden Klassenvariablen, *MAX_POINTS_ON_MAP* und *LIMIT_POINTS_ENABLED*, wurden deshalb eingeführt, da es zu Performanceengpässen kommt, wenn mehrere Tausend GPS Koordinaten miteinander verbunden werden müssen. Das Problem tritt deshalb auf, da bei jeder Kartenverschiebung oder bei der Änderung des Zoomlevels alle Verbindungen neu gezeichnet werden und so die Interaktion nicht mehr flüssig abläuft[17]. Die Variable *MAX_POINTS_ON_MAP* legt dabei fest, wie viele Punkte einer Fahrt tatsächlich verwendet werden sollen (in diesem Beispiel hier 500). Mit der Variable *LIMIT_POINTS_ENABLED* kann diese mengenmässige Beschränkung aktiviert (*true*) oder deaktiviert (*false*) werden. In der Zeile *„int everyPoint . . . “* wird die Anzahl aller Punkte der Fahrt durch die *LIMIT_POINTS_ENABLED* geteilt. Der gerundete Wert der Variable *everyPoint* besagt, die wievielten Punkte jeweils angezeigt werden sollen. Falls die Fahrt beispielsweise insgesamt 2'500 GPS Koordinaten besitzt, würde nur jeder fünfte Punkt verwendet werden, damit schlussendlich die Karte nicht mehr als 500 Punkte enthält. Im *if*-Statement in Auflistung 3.1 werden dazu, falls *LIMIT_POINTS_ENABLED* den Wert *true* besitzt, die Datensätze des Datenbank Cursors entsprechend übersprungen.

[17]Dies v.a. im Android Emulator. Auf einem HTC Desire Android Telefon läuft die Interaktion einigermassen flüssig ab.

```
[...]
private static final double MAX_POINTS_ON_MAP = 500.0d;
private static final boolean LIMIT_POINTS_ENABLED = true;
[...]
int everyPoint = (int) Math.ceil((double) (count / MAX_POINTS_ON_MAP));
[...]
if (LIMIT_POINTS_ENABLED) {
    for (int j = 0; j < everyPoint; j++) {
        mCursor.moveToNext();
    }
} else {
    mCursor.moveToNext();
}
[...]
```

Auflistung 3.1: Auszug aus *ShowMapActivity.java*

3.5.3.4 Fahrtenvergleich und Chart Export

Die reale Fahrt und allfällige Simulationen können im *Comparison Chart*, auswählbar in der Projektübersicht (siehe Abbildung 3.11a), betrachtet werden. Dabei wird die Activity *ComparisonChartActivity* gestartet, in der zuerst alle vorhandenen Daten sowohl des Projektes als auch der Simulationen in den Speicher geladen werden. So sind die Daten schnell vorhanden, wenn der/die BenutzerIn die Chart-Optionen ändert. Abbildung 3.13a zeigt den Chart, der beim Öffnen der Activity angezeigt wird. Die in Abbildung 3.13a dargestellten Ausreisser beruhen auf GPS Messungenauigkeiten (mehr dazu im Abschnitt 3.5.3.5).

Der Chart bildet ein zweidimensionales Koordinatensystem ab. Er kann bewegt, vergrössert oder verkleinert werden. Zudem steht ein Button zur Verfügung („1:1“), der den Chart wieder in den ursprünglichen Zustand versetzt und den Zoomlevel so setzt, damit alle Werte sichtbar sind.

Die X-Achse repräsentiert jeweils den zurückgelegten Weg in Metern. Es gibt sowohl eine linke, als auch eine rechte Y-Achse. Die Werte der Y-Achsen lassen sich mit dem Dialog *Select Data* festlegen (siehe Abbildung 3.13b). Die linke Y-Achse kann entweder Werte zum Totalverbrauch (in kW), zu den einzelnen Kräften (in N), zur Geschwindigkeit (in km/h) oder zur Höhe über Meer (in m) annehmen. Die rechte Y-Achse kann keine Werte oder die gleichen Werte wie die linke Y-Achse (ausser den Kräften) annehmen.

Die reale Fahrt, die bei der Projekterstellung (siehe Abschnitt 3.5.3.1) eingelesen wurde, wird als *Original Run* bezeichnet. Mit *Select Runs* können diejenigen Läufe (realer Lauf und Simulationen) selektiert werden, die im Chart miteinander verglichen werden sollen (siehe Abbildung 3.14a).

Die Werte im Koordinatensystem werden mit unterschiedlichen Farben gezeichnet. In der Legende sind alle Werte und deren Läufe aufgelistet. Wie in Abbildung 3.14b gezeigt, wurde für den Chart der *Original Run* (reale Fahrt) und auch die Simulation *Simul 1* ausgewählt. Es werden sowohl der Luftwiderstand (*F_aero*, linke Y-Achse) als auch die Höhe über Meer (*Altitude*, rechte Y-Achse) dargestellt (zuvor unter *Select Data* so

(a) Initial Chart (b) Datenauswahl

Abbildung 3.13: Comparison Chart und Auswahl der Daten

(a) Auswahl der Läufe (b) Chart mit realem Lauf und einer Simulation

Abbildung 3.14: Comparison Chart und Auswahl der Daten

ausgewählt). Für die reale Fahrt und die Simulation gibt es unterschiedliche Werte zum Luftwiderstand, weshalb sie auch einzeln im Chart gezeichnet werden (*Original Run F_*

aero und *Simul 1 F_aero*). Da die Höhe über Meer von der Simulation nicht geändert werden kann, wird dieser Wert jeweils nur einmal im Chart abgebildet.

Der aktuelle Chart-Ausschnitt kann via *Export Chart* als PNG-Grafikdatei auf die SD-Karte exportiert werden.

Mit den von der Empa zur Verfügung gestellten Testdaten (siehe Anhang A) sind knapp 3'000 Messreihen eingelesen worden. Pro Linie im Chart werden bei der Betrachtung der Testdaten 3'000 Punkte eingezeichnet. Bei jeder Änderung des Charts werden all diese Punkte neu gezeichnet, was im Emulator zu Performanceengpässen führt, d.h. die Interaktion läuft nicht mehr flüssig ab. Auf einem HTC Desire Telefon, beispielsweise, läuft es jedoch um einiges flüssiger ab.

Die Chart Implementierung nutzt eine quelloffene Chart-API der Firma *The 4ViewSoft Company* (The 4ViewSoft Company, 2011).

3.5.3.5 Berechnung der physikalischen Kräfte

Bei der Projekt- und Simulationserstellung und -bearbeitung werden die in Kapitel 2 beschriebenen physikalischen Kräfte berechnet. Dies geschieht mit Werten, die der Inputdatendatei entnommen werden, mit Werten, die der/die BenutzerIn bei der Parameterbearbeitung, dem Spalten- und dem Voltagemapping eingegeben hat, und mit Werten, die als Konstanten im Quellcode definiert wurden.

Zur Berechnung der Kräfte wird auf GPS Messdaten der Sensorenbox zurückgegriffen. Je nach Konstellation (Gerätetyp und Anzahl sichtbarer Satelliten) können bei GPS Daten Messungenauigkeiten von mehreren Metern auftreten (Mansfeld, 2010). Solche Messfehler bzw. -ungenauigkeiten wirken sich natürlich auf die Berechnungen aus. Die Abbildung 3.13a zeigt beispielsweise diverse Ausreisser, die auf solchen Messungenauigkeiten beruhen. Beispielsweise könnte ein extrem grosser Steigungswinkel, der in Tat und Wahrheit nur aufgrund von Messfehlern so ausfällt, zu solchen Ausreissern führen, da zur Überwindung der grossen Steigung massive Kraft aufgewendet werden müsste[18]. Bei der Implementierung wurde darauf verzichtet, solche Ausreisser anhand Plausibilitätschecks und entsprechenden Justierungen zu verhindern. Ein Ansatz dazu könnte der Douglas-Peucker-Algorithmus sein, der solche Ausreisser identifizieren und eliminieren könnte[19]. Wie weiter unten gezeigt wird, können Messfehler auch dazu führen, dass unbrauchbare Daten vorliegen, mit denen gar keine Berechnungen durchgeführt werden können.

Die Berechnung der physikalischen Kräfte geschieht in der Klasse *Forces* im Paket *ebike.physics*. Es müssen folgende, in Kapitel 2 bereits erwähnten, physikalische Kräfte berechnet werden: der Luft-, der Rad-, der Steigungs- und der Beschleunigungswiderstand. Tabelle 2.1 aus dem Abschnitt 2.3.2 zeigt alle erforderlichen Werte. Ebenfalls wird die Antriebskraft berechnet, die zur Überwindung der genannten Widerstände benötigt wird (siehe Abschnitt 2.2.5).

[18]Im gezeigten Beispiel ist die ungenaue Messung der GPS Geschwindigkeit für die Ausreisser verantwortlich, die dann die Berechnung der Beschleunigung drastisch verfälscht.

[19]Zu diesem Algorithmus finden sich im Internet diverse Information, u.a. hier als C++ Implementierung: `http://www.codeproject.com/KB/recipes/dphull.aspx`.

Der **Luftwiderstand** (siehe Gleichung (2.4)) wird anhand der Luftdichte, der Relativgeschwindigkeit, des Luftwiderstandskoeffizienten und der Stirnfläche berechnet. Die beiden letzt genannten Werte werden vom/von der BenutzerIn bei der Parametereingabe festgelegt (siehe Abschnitt 3.5.3.1). Die Relativgeschwindigkeit steht als Inputwert der eingelesenen Datendatei zur Verfügung (GPS Velocity). Für die Luftdichte stehen keine Werte aus der Inputdatendatei zur Verfügung. Deshalb wird für die Luftdichte ein konstanter Wert von $1.293 kg/m^3$ angenommen (siehe Gleichung (2.7)). Auflistung 3.2 zeigt die Implementierung der Berechnung des Luftwiderstandes.

```
[...]
private static final double AIR_DENSITY = 1.293d;
[...]
public static double getAerodynamicResistanceForce(double airDensity,
    double airDragCoeff, double frontArea, double velocity,
    int velocityType) {
    [...]
    return (airDensity / 2) * airDragCoeff * frontArea *
        Math.pow(velocity, 2);
}
[...]
```

Auflistung 3.2: Berechnung des Luftwiderstandes; Auszug aus *Forces.java*

Der **Radwiderstand** (siehe Gleichung (2.11)) wird anhand des Gewichtes (Motorrad und FahrerIn), der Erdbeschleunigung, des Rollwiderstandskoeffizienten und des Kosinus des Steigungswinkels berechnet. Die Gewichtsangaben und der Rollwiderstandsbeiwert werden bei den Parametern manuell festgelegt. Der Normwert der Erdbeschleunigung beträgt $9.80665 m/s^2$ (DMK und DPK, 1995). Der Steigungswinkel wird mit Hilfe der Distanz zwischen zwei aufeinanderfolgenden Messreihen, der Höhe über Meer der GPS Koordinaten und des Satzes von Pythagoras berechnet. Die Berechnung des Steigungswinkels wird nun detailliert erklärt.

Die Angabe bezüglich der Höhe über Meer wird in der Inputdatendatei pro Zeile und daher auch pro GPS Koordinate geliefert (GPS Altitude).

Die Distanz zwischen zwei aufeinanderfolgenden Messreihen wird mit Hilfe der Methode *getDistance* der Klasse *Distance* aus dem Paket *ebike.gps* berechnet. Die Distanz wird anhand folgender Gleichung berechnet (DMK und DPK, 1995):

$$s = \frac{1}{2} * a * t^2 + v_0 * t \begin{cases} s: & \text{Zurückgelegte Strecke (m)} \\ a: & \text{Beschleunigung (m/s}^2\text{)} \\ t: & \text{Verwendete Zeit (s)} \\ v_0: & \text{Geschwindigkeit zum vorherigen Zeitpunkt (m/s)} \end{cases} \tag{3.1}$$

Die Gleichung (3.1) geht davon aus, dass die Beschleunigung zwischen den Messreihen konstant ist. Gemäss den von der Empa zur Verfügung gestellten Messdaten (siehe Anhang A) sammelt die Sensorenbox mit einer Taktfrequenz von wenigen Millisekunden. Aufgrund der hohen Frequenz führt die Annahme einer konstanten Beschleunigung nicht zu grossen Berechnungsungenauigkeiten. Sollte die Frequenz jedoch verkleinert werden (z.B. Messdaten werden alle 2 Sekunden gesammelt), würde der Fehler zunehmen. Auflistung 3.3 zeigt die implementierte Berechnung in der Methode *getDistance*.

```
[...]
public static double getDistance(double prevTime, double prevVelocity,
    double currTime, double currVelocity) {

    // Get meters per second
    prevVelocity = PhysicsHelper.getMetersPerSecond(prevVelocity);
    currVelocity = PhysicsHelper.getMetersPerSecond(currVelocity);

    // Delta of the time
    double deltaTime = currTime - prevTime;

    // Get acceleration
    double a = PhysicsHelper.getAcceleration(prevVelocity,
        currVelocity, prevTime, currTime,
        PhysicsHelper.VELOCITY_IN_MPS, PhysicsHelper.TIME_IN_S);

    return (0.5 * a * Math.pow(deltaTime, 2)) +
        (prevVelocity * deltaTime);
}
[...]
```

Auflistung 3.3: Distanzberechnung; Auszug aus *Distance.java*

Der Steigungswinkel kann nun anhand des Satzes von Pythagoras berechnet werden (DMK und DPK, 1995). Auflistung 3.4 zeigt die Methode *getAngle* der Klasse *SlopeAnlge*, die im Paket *ebike.gps* liegt. Dabei hat die Verwendung der zur Verfügung gestellten Testdaten folgendes Problem aufgezeigt: wegen ungenauer GPS-Daten kann die Situation auftreten, bei der der Abstand zwischen den beiden GPS Punkten kleiner ist als der Höhenunterschied derselben. Das ist mathematisch gesehen, nicht möglich. Da so der Arkussinus nicht berechnet werden kann, wurde bei der Implementierung entschieden, dass in diesen Fällen ein Steigungswinkel von 0 Grad angenommen wird. Zudem zeigt die Verwendung des Satzes von Pythagoras folgender Kompromiss auf: es wird angenommen, dass die Strecke zwischen zwei Messreihen eine Gerade ist, d.h. ohne Krümmung. Falls die Frequenz, mit der die Sensorenbox Messdaten sammelt, während der Messung konstant ist (z.B. alle 100 Millisekunden wird gemessen), wird die Strecke mit zunehmender Geschwindigkeit grösser. Und je grösser die Strecke desto grösser die Wahrscheinlichkeit, dass die Verwendung des Satzes von Pythagoras zu einer ungenauen Berechnung des Radwiderstandes führt.

```
[...]
public static double getAngle(double aAlt, double bAlt, double distance
    ) {

    // Catch division by zero (a)
    if (distance != 0.0d) {

        // Catch GPS measure failure (b)
        if (distance > Math.abs(bAlt - aAlt)) {

            // Return the arc sinus of a / c
            return Math.asin((bAlt - aAlt) / distance);
        }
    }
```

```
    /*
     * Reaching this point, will return 0.0d, what could have been
     * produced by one of the following reasons:
     *
     * (a) If both points are exactly the same, means, the vehicle
     * didn't move (distance == 0.0d ) and therefore there is no
     * slope angle between them.
     *
     * (b) If there is a GPS measure failure or the measure data is not
     * exactly enough: the distance between two points cannot be
        smaller
     * than the difference in the altitude of these points. Because we
     * cannot calculate this, 0.0d will be returned, what is in fact
     * also not correct.
     */
    return 0.0d;
}
[...]
```

Auflistung 3.4: Berechnung des Steigungswinkels; Auszug aus *SlopeAnlge.java*

Auflistung 3.5 zeigt die Implementierung der Berechnung des Radwiderstandes.

```
[...]
private static final double GRAVITY_ACCELERATION = 9.80665d;
[...]
public static double getRollingResistanceForce(double driverWeight,
   double bikeWeight, double rollCoeff, double alpha) {

    return (driverWeight + bikeWeight) * GRAVITY_ACCELERATION *
       rollCoeff * Math.cos(alpha);
}
[...]
```

Auflistung 3.5: Berechnung des Radwiderstandes; Auszug aus *Forces.java*

Der **Steigungswiderstand** (siehe Gleichung (2.12)) setzt sich aus dem Gesamtgewicht, der Erdbeschleunigung und dem Sinus des Steigungswinkels zusammen. Das Gesamtgewicht wird vom/von der BenutzerIn bei den Parametern eingeben. Die Erdbeschleunigung und die Berechnung des Steigungswinkels wurden eben erklärt. Auflistung 3.6 zeigt die Implementierung des Steigungswiderstandes. Bei der Berechnung des Steigungswiderstandes gelten die gleichen Einschränkungen bezüglich der Ungenauigkeit des Steigungswinkels wie bei der Ermittlung des Radwiderstandes. Auch hier wiederum gilt: da die Geschwindigkeitsmessung auf GPS-Daten beruht, kann es zu Ungenauigkeiten bzw. Verfälschungen in der Berechnung der Beschleunigung kommen.

```
[...]
private static final double GRAVITY_ACCELERATION = 9.80665d;
[...]
public static double getSlopeResistanceForce(double driverWeight,
    double bikeWeight, double alpha) {

    return (driverWeight + bikeWeight) * GRAVITY_ACCELERATION *
        Math.sin(alpha);
}
```

```
[...]
```

Auflistung 3.6: Berechnung des Steigungswiderstandes; Auszug aus *Forces.java*

Der **Beschleunigungswiderstand** (siehe Gleichung (2.18)) setzt sich aus dem Massenfaktor, dem Gesamtgewicht und der Beschleunigung zusammen. Der/die BenutzerIn legt bei der Parametereingabe den Massenfaktor und das Gesamtgewicht fest. Die Beschleunigung wird anhand der Differenz der Relativgeschwindigkeit und der benötigten Zeit des aktuellen und vorhergehenden Messpunktes bestimmt (siehe Gleichung (2.13)). Auflistung 3.7 zeigt die Implementierung des Beschleunigungswiderstandes.

```
[...]
public static double getAccelerationResistanceForce(
    double driverWeight, double bikeWeight, double inertiaFactor,
    double acceleration) {

    return ((inertiaFactor * bikeWeight) + driverWeight) *
        acceleration;
}
[...]
```

Auflistung 3.7: Berechnung des Beschleunigungswiderstandes; Auszug aus *Forces.java*

Schliesslich lässt sich die **Antriebskraft** gemäss den Gleichungen (2.19) und (2.20) aus der Summe der eben beschriebenen Widerstände berechnen.

3.5.4 Datenmodell und Datenbankimplementierung

Die Applikation verwendet eine SQLite Datenbank. In der Datenbank werden einerseits die eingelesenen Daten gespeichert, andererseits die vom/von der BenutzerIn getätigten Angaben festgehalten. Das für diese Applikation verwendete Datenmodell wird in Abbildung 3.15 gezeigt.

Alle Tabellen enthalten einen selbstinkrementierenden Primärschlüssel *_id*. Die einzelnen Tabellen, deren Relationen und zusätzlichen Feldern werden nun beschrieben. Die Tabellen sind im Paket *ebike.database* implementiert. Nachfolgende Referenzierungen zu Java Klassen beziehen sich auf dieses Paket, falls nicht anders erwähnt.

Die Tabelle **project** beinhaltet das Feld *name*, in das der Projektname geschrieben wird. Ausserdem wird der Pfad zur Datei festgehalten (*file_path*). *start_row* gibt diejenige Zeile an, ab der in der Inputdatendatei mit dem Einlesen der Daten begonnen werden kann. Die Felder *a00_voltage_offset*, *a00_voltage_scale* und *a01_voltage_offset* stellen die Werte des Voltagemappings dar. Schliesslich wird bei der Projekterstellung oder -änderung jeweils die aktuelle Zeit als UNIX Zeitstempel in das Feld *timestamp* geschrieben. Die Tabellenimplementierung findet sich in der Klasse *ProjectTbl*.

Bei der Implementierung sieht man, dass Felder, die Dezimalzahlen als Datentyp repräsentieren, als *real* und nicht als *double*, wie in Abbildung 3.15 gezeigt, erstellt werden. In SQLite werden Dezimalzahlen mit dem Datentyp *real* gespeichert, was einen 8-Byte langen Floating Point Wert darstellt (SQLite, 2011a). In Java gibt es für Dezimalzahlen zwei

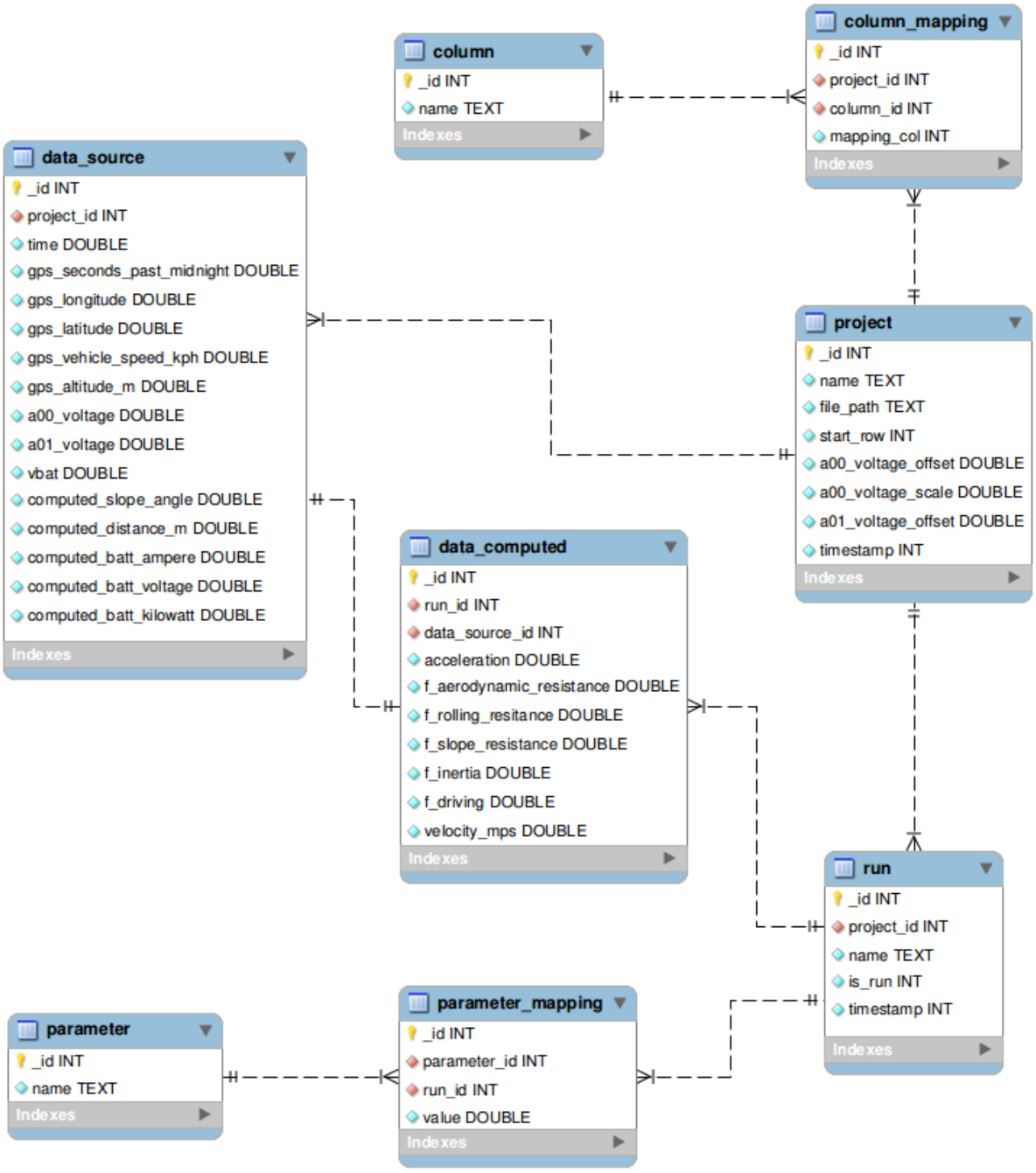


Abbildung 3.15: Datenmodell der Applikation

primitive Datentypen: *float* (4 Bytes) oder *double* (8 Bytes). Bei der Implementierung stellen deshalb *real*-Werte in SQLite *double*-Werte in Java dar.[20]

Die Tabelle **run** beinhaltet den Namen des Laufs (Feld *name*). Das Feld *is_run* ist ein

[20] In SQLite wird ein gänzlich anderes Konzept bezüglich Datentypen verfolgt als bei anderen relationalen Datenbanksystemen. Gewöhnlich haben Tabellenfelder starre Datentypen als Basis (also z.B. Integer, Blob, Text, etc.). Bei SQLite wird der Datentyp erst zur Laufzeit dynamisch bestimmt. Mehr dazu unter `http://www.sqlite.org/datatype3.html`.

Boolean[21], der besagt, ob es sich beim Run um eine Simulation (*is_run = 0*) oder um den Lauf der realen Fahrt (*is_run = 1*) handelt. Das Feld *timestamp* beinhaltet von jedem Lauf den UNIX Zeitstempel der letzten Änderung. Der Fremdschlüssel *project_id* signalisiert, dass zu jedem Lauf genau ein Projekt gehört. Die Tabelle wird in der Klasse *RunTbl* implementiert.

Die vorliegende Applikation basiert auf der Android SDK-Version 2.2, die wiederum SQLite in der Version 3.6.22 benutzt (siehe Abschnitt 3.5.2). SQLite implementiert ab der Version 3.6.19 das Konzept der Fremdschlüssel (SQLite, 2011b). Auflistung 3.8 zeigt anhand der Klasse *RunTbl*, wie ein Fremdschlüssel in SQLite deklariert wird.

```
[...]
public static final String COL_ID = "_id";
public static final String COL_PROJECT_ID = "project_id";
public static final String COL_NAME = "name";
public static final String COL_IS_RUN = "is_run";
public static final String COL_TIMESTAMP = "timestamp";
[...]
public static final String SQL_CREATE =
    "CREATE TABLE " + TABLE_NAME + " ("
    + COL_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
    + COL_PROJECT_ID + " INTEGER NOT NULL, "
    + COL_NAME + " TEXT NOT NULL, "
    + COL_IS_RUN + " INTEGER NOT NULL, "
    + COL_TIMESTAMP + " INTEGER NOT NULL, "
    + "FOREIGN KEY (" + COL_PROJECT_ID + ") REFERENCES "
            + ProjectTbl.TABLE_NAME + "(" + ProjectTbl.COL_ID + "));";
[...]
```

Auflistung 3.8: Deklaration von Fremdschlüssel in SQLite; Auszug aus *RunTbl.java*

Bei SQLite ist die Fremdschlüsselunterstützung standardmässig deaktiviert. Deshalb wird sie beim Öffnen der Datenbank jeweils aktiviert. Dies geschieht in der Methode *open* der Klasse *DatabaseAdapter* und wird in Auflistung 3.9 gezeigt.

```
[...]
public DatabaseAdapter open() throws android.database.SQLException {
    mDbHelper = new DatabaseHelper(mCtx);
    mDb = mDbHelper.getWritableDatabase();

    // Enable foreign key constraints
    if (!mDb.isReadOnly()) {
        mDb.execSQL("PRAGMA foreign_keys=ON;");
    }

    return this;
}
[...]
```

Auflistung 3.9: Aktivierung der Fremdschlüssel; Auszug aus *DatabaseAdapter.java*

[21] Für Booleans gibt es keinen eigenständigen Wert in SQLite, sondern es wird ein Integer verwendet, der entweder den Wert 0 (für *false*) oder 1 (für *true*) annimmt (SQLite, 2011a).

Die Tabelle **parameter** enthält die für jeden Lauf benötigten Parameter (wie z.B. das Gewicht des Motorrades, der Luftwiderstandskoeffizient, etc.). Ein Parameter zeichnet sich durch seinen Primarschlüssel (*_id*) und seinen Namen (*name*) aus. Die Parameter werden beim erstmaligen Start der Applikation in die Datenbank geschrieben und zwar mit Hilfe der Methode *insertInitialData* in der Klasse *DatabaseAdapter*. Die Implementierung der Tabelle befindet sich in der Klasse *ParameterTbl*.

Die Tabelle **parameter_mapping** ist – wie es der Name bereits verrät – für das Mapping der Parameter mit einem Lauf zuständig. Dabei werden als Fremdschlüssel sowohl der Lauf als auch der spezifische Parameter festgehalten (Felder *run_id* und *parameter_id*). Zusätzlich dazu interessiert natürlich der Wert, der für einen bestimmten Lauf und für einen bestimmten Parameter vom/von der Android BenutzerIn bei der Projekt- und Simulationserstellung oder -bearbeitung festgelegt wurde (Feld *value*). Die Tabelle wird mit Hilfe der Klasse *ParameterMappingTbl* erstellt.

Bei der Projekterstellung kann der/die BenutzerIn das Spaltenmapping vornehmen (siehe Abschnitt 3.5.3.1). Dafür braucht es die Tabellen **column** und **column_mapping**. Die Tabelle *column* enthält die Namen der von der Inputdatendatei zwingend benötigten Spalten (siehe Abschnitt 3.5.3.1), die mit Hilfe der bereits erwähnten Methode *insertInitialData* erstellt werden. Die Tabelle *column_mapping* weist pro Spalte (*column_id*) und pro Projekt (*project_id*) eine Spaltennummer (*mapping_col*) zu. So ist es pro Projekt möglich, verschiedenartig aufgebaute Inputdatendateien einzulesen. Die beiden Tabellen werden in den Klassen *ColumnTbl* und *ColumnMappingTbl* implementiert.

In die Tabelle **data_source** werden die eingelesenen Daten geschrieben. Es werden nicht nur die ursprünglichen Daten, sondern auch bereits bearbeitete Daten (wie z.B. der Steigungswinkel im Feld *computed_slope_angle*) gespeichert, da solche Werte nicht durch Simulationen verändert werden können und sie thematisch gesehen zu den Inputwerten gehören. Der Fremdschlüssel *project_id* stellt dabei sicher, dass jedem Projekt nur eine Datenquelle zugrunde liegen kann. Die Klasse *DataSourceTbl* zeigt die Tabellenimplementierung.

Die Tabelle **data_computed** ist ein Bindeglied zwischen den eingelesenen Daten (Fremdschlüssel *data_source_id*) und dem realen Lauf bzw. einer Simulation (Fremdschlüssel *run_id*). In ihr werden die berechneten physikalischen Kräfte (Felder beginnend mit *f_*), die Relativbeschleunigung (*acceleration*) und die Relativgeschwindigkeit (*velocity_mps*) gespeichert. Die Tabelle wird mit Hilfe der Klasse *DataComputedTbl* gebildet.

Wie in Abschnitt 3.5.1 erwähnt, soll aus Gründen der Performance auf eine komplexe Datenbankschicht verzichtet werden. Als Datenbankzugriffsschicht agieren deshalb lediglich zwei Klassen: *DatabaseHelper* und *DatabaseAdapter*. In *DatabaseHelper* finden sich Methoden, die sowohl für die Erstellung als auch für die Veränderung des Datenbankschemas verantwortlich sind. Für die eigentlichen Operationen auf der Datenbank ist die Klasse *DatabaseAdapter* zuständig. Sie dient als Schnittstelle zwischen der Datenbank und den anderen Klassen, um Abfrage-, Einfüge-, Lösch- und Aktualisierungsvorgänge vorzunehmen. Wie bereits erwähnt, werden in ihr auch die Tabellen *ColumnTbl* und *ParameterTbl* initialisiert.

Kapitel 4

Zusammenfassung und Ausblick

Die Empa St. Gallen hat eine Sensorenbox entwickelt, mit der es möglich ist, während der Fahrt mit einem Elektromotorrad Daten zu erheben und zu sammeln. Die Daten beinhalten einerseits Werte eines GPS-Gerätes, das Positionssignale von Satelliten empfängt und auswertet, andererseits werden Messungen an der Batterie, die den Elektromotor mit Strom versorgt, vorgenommen.

Die Fahrt eines Motorrades wird von verschiedenen physikalischen Kräften beeinflusst. Um ein Motorrad zu bewegen oder in Bewegung zu halten, müssen diese Kräfte durch den Antrieb des Motors überwunden werden.

Anhand der erhobenen Fahrtdaten können sowohl die reale Fahrt mit ihren physikalischen Kräften berechnet werden, als auch mit geänderten Parametern eine virtuelle Fahrt simuliert werden. Die Arbeit ergänzt auf diese Weise die Entwicklung der Sensorenbox, indem die gesammelten Daten zusammen mit den Berechnungen der physikalischen Kräfte mit Hilfe einer Software verarbeitet und ausgewertet werden.

Die physikalischen Kräfte (in *Newton* gemessen), die auf ein Motorrad auf geradliniger Fahrt wirken, lauten: der Luftwiderstand, der Radwiderstand, der Steigungswiderstand, der Beschleunigungswiderstand und die Antriebskraft. Der Luftwiderstand steigt quadratisch zur Relativgeschwindigkeit und hängt von aktuellen Umweltbedingungen, der Strömungsgüte und der Stirnfläche des Motorrades ab. Der Radwiderstand setzt sich aus der Normalkraft (Gewicht, Erdbeschleunigung und Steigungswinkel) und dem Rollwiderstandskoeffizienten zusammen. Der Steigungswiderstand bildet sich anhand des Gewichtes, der Erdbeschleunigung und des Steigungswinkels. Bergauf nimmt der Steigungswiderstand positive, bergab negative Werte an. Bergab wirkt der vermeintliche Widerstand daher als zusätzliche Antriebskraft. Bei horizontaler Fahrt wirkt kein Steigungswiderstand. Bei einer Änderung des Bewegungszustandes (Richtung bzw. Geschwindigkeit) eines Körpers muss der Trägheits- bzw. Beschleunigungswiderstand überwunden werden. Der Beschleunigungswiderstand lässt sich mit Hilfe der Beschleunigung, des Gewichtes und des Massenfaktors berechnen. Die Antriebskraft, die am Rad wirken muss, um die erwähnten Widerstände zu überwinden, bildet sich anhand der Summe dieser Widerstände. Zusätzlich zur Antriebskraft interessiert vor allem auch die Antriebsleistung. Die Antriebsleistung ist das Produkt der Antriebskraft multipliziert mit der Relativgeschwindigkeit und wird in der Einheit *Watt* ausgewiesen.

Mit dem System Dynamics Ansatz wird das Simulationsmodell gebildet. Die physikalischen Kräfte werden dabei in ein Ursache- und Wirkungsverhältnis gesetzt. Allfällige Rückkoppellungen spielen beim System Dynamics Ansatz ebenfalls eine wichtige Rolle und finden sich auch im hier vorliegenden Modell wieder.

Mit den Erkenntnissen aus der Erarbeitung der physikalischen Kräfte und deren Simulationsmodell wird die Applikation entwickelt. Da die Möglichkeit bestehen soll, dass die Software auf mobilen Geräten zum Einsatz kommen kann, wurde als Framework die Android Plattform gewählt. Die Implementierung deckt diverse Anwendungsfälle ab. So kann der/die BenutzerIn ein Projekt erstellen, bearbeiten oder entfernen. Für jedes Projekt werden Daten der Sensorenbox in Form einer CSV-Datei eingelesen. Der/die BenutzerIn muss für die Berechnung der Kräfte die Inputdaten mit Parametern ergänzen. Ein Projekt stellt dabei eine reale Fahrt mitsamt ihren Parametern und Inputdaten dar. Eine reale Fahrt kann mit anderen Parametern simuliert werden. Simulationen können zudem bearbeitet oder gelöscht werden. Das Projekt bzw. die Fahrt kann in Google Maps betrachtet werden. Schliesslich ermöglicht die Applikation dem/der AnwenderIn die tatsächliche Fahrt mit ihren Simulationen in einem Chart zu vergleichen und diesen als PNG-Datei zu exportieren.

Die Applikation beruht auf der Android SDK-Version 2.2. Damit ist sie mit über 83.5% der aktuell verwendeten Android Geräte kompatibel. Die Applikation besteht aus mehr als 3'000 Zeilen Java, aus mehr als 600 Kommentarzeilen und aus ein paar Hundert Zeilen XML.

Die Applikation soll als erster Schritt zur Verwendung der Inputdaten verstanden werden. Sie wurde so aufgebaut, dass zukünftige Änderungen und Erweiterungen problemlos möglich sein sollten. Die hier vorgenommene Implementierung löst das sogenannte Vorwärtsproblem. Durch die Entwicklung entsprechender Algorithmen kann die Applikation dahingehend erweitert werden, sodass auch das Rückwärtsproblem gelöst werden kann.

Anhang A

Auszug der Inputdatendatei

```
Time,GPS Track made good,GPS Longitude,GPS Seconds past midnight UTC,GPS Vehicle speed,GPS Vehicle speed,GPS Altitude,GPS Altitude,GPS Satellite count,GPS LatG,GPS Latitude,A00 voltage,A01 voltage,GPS Distance,GPS Distance,GPS Distance,GPS Distance,GPS LongG,Vbat,Internal VertG,Internal LongG,Internal LatG,
s,d,d,s,MPH,KPH,ft,m,sats,g,d,V,V,ft,miles,km,% 1/4 mile,g,V,g,g,g,
     0.000,108.25, +9.3424169,57523.427,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,0.000,0.000,  0.000,  0.000,  0.000,  0.0,+0.00, 0.00,+1.00,+0.00,-0.00,
     0.062,108.25, +9.3424169,57523.489,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,0.000,0.000,  0.000,  0.000,  0.000,  0.0,+0.00, 0.00,+1.00,+0.00,-0.00,
     0.103,108.25, +9.3424152,57523.530,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.521,3.737,  0.000,  0.000,  0.000,  0.0,+0.00,11.86,+1.06,+0.04,-0.08,
     0.375,108.25, +9.3424135,57524.200,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.521,3.737,  0.000,  0.000,  0.000,  0.0,+0.00,11.86,+1.06,+0.04,-0.08,
     0.475,108.25, +9.3424135,57524.300,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.524,3.738,  0.000,  0.000,  0.000,  0.0,+0.00,11.85,+1.12,+0.07,-0.17,
     0.575,108.25, +9.3424135,57524.400,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.527,3.739,  0.000,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     0.675,108.25, +9.3424135,57524.500,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.528,3.741,  0.033,  0.000,  0.000,  0.0,+0.00,11.85,+1.18,+0.11,-0.26,
     0.775,108.25, +9.3424135,57524.600,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.526,3.740,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     0.875,108.25, +9.3424135,57524.700,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.528,3.735,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     0.976,108.25, +9.3424135,57524.801,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.529,3.738,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.17,+0.11,-0.25,
     1.076,108.25, +9.3424135,57524.901,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.523,3.741,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.17,+0.11,-0.25,
     1.176,108.25, +9.3424135,57525.001,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.526,3.737,  0.066,  0.000,  0.000,  0.0,+0.00,11.87,+1.18,+0.11,-0.26,
     1.276,108.25, +9.3424135,57525.101,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.521,3.746,  0.066,  0.000,  0.000,  0.0,+0.00,11.87,+1.18,+0.11,-0.26,
     1.286,108.25, +9.3424135,57525.111,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.521,3.746,  0.066,  0.000,  0.000,  0.0,+0.00,11.87,+1.18,+0.11,-0.26,
     1.377,108.25, +9.3424135,57525.202,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.523,3.737,  0.066,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     1.477,108.25, +9.3424135,57525.302,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.527,3.733,  0.066,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     1.578,108.25, +9.3424135,57525.403,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.526,3.739,  0.098,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     1.678,108.25, +9.3424135,57525.503,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.526,3.738,  0.098,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     1.778,108.25, +9.3424135,57525.603,  0.0,  0.0, +2139.1, +652.0, 6,+0.00,+47.4128945,2.526,3.737,  0.098,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.12,-0.25,
     1.779,108.25, +9.3424152,57525.000,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.526,3.737,  0.098,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.12,-0.25,
     1.879,108.25, +9.3424152,57525.100,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.523,3.743,  0.098,  0.000,  0.000,  0.0,+0.00,11.84,+1.18,+0.12,-0.25,
     1.979,108.25, +9.3424152,57525.200,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.526,3.738,  0.066,  0.000,  0.000,  0.0,+0.00,11.86,+1.17,+0.11,-0.25,
     2.080,108.25, +9.3424152,57525.301,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.528,3.741,  0.066,  0.000,  0.000,  0.0,+0.00,11.85,+1.18,+0.11,-0.25,
     2.180,108.25, +9.3424152,57525.401,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.524,3.741,  0.066,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     2.280,108.25, +9.3424152,57525.501,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.527,3.738,  0.066,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     2.381,108.25, +9.3424152,57525.602,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.524,3.744,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     2.481,108.25, +9.3424152,57525.702,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.527,3.739,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     2.484,108.25, +9.3424152,57525.705,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.527,3.739,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     2.582,108.25, +9.3424152,57525.803,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.526,3.740,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.17,+0.11,-0.25,
     2.682,108.25, +9.3424152,57525.903,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.527,3.739,  0.000,  0.000,  0.000,  0.0,+0.00,11.86,+1.17,+0.11,-0.26,
     2.782,108.25, +9.3424152,57526.003,  0.0,  0.0, +2139.1, +652.0, 7,+0.00,+47.4128961,2.522,3.745,  0.000,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     2.783,108.25, +9.3424169,57525.800,  0.0,  0.0, +2138.8, +651.9, 6,+0.00,+47.4128978,2.522,3.745,  0.000,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     2.883,108.25, +9.3424169,57525.900,  0.0,  0.1, +2138.8, +651.9, 6,+0.00,+47.4128978,2.524,3.739,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     2.983,108.25, +9.3424185,57526.000,  0.1,  0.1, +2138.8, +651.9, 6,+0.00,+47.4128995,2.524,3.744,  0.098,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     3.084,108.25, +9.3424185,57526.101,  0.1,  0.2, +2138.8, +651.9, 6,+0.00,+47.4128995,2.527,3.740,  0.131,  0.000,  0.000,  0.0,+0.00,11.85,+1.18,+0.11,-0.25,
     3.184,108.25, +9.3424202,57526.201,  0.2,  0.3, +2138.8, +651.9, 6,+0.00,+47.4129011,2.529,3.743,  0.197,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     3.187,108.25, +9.3424202,57526.600,  0.2,  0.3, +2138.5, +651.8, 6,+0.00,+47.4129011,2.529,3.743,  0.197,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     3.288,108.25, +9.3424202,57526.701,  0.2,  0.3, +2138.5, +651.8, 6,+0.00,+47.4129011,2.526,3.739,  0.164,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     3.389,108.25, +9.3424202,57526.802,  0.2,  0.3, +2138.1, +651.7, 6,+0.00,+47.4129011,2.527,3.741,  0.131,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     3.490,108.25, +9.3424202,57526.903,  0.2,  0.3, +2138.1, +651.7, 6,+0.00,+47.4129011,2.527,3.739,  0.066,  0.000,  0.000,  0.0,+0.00,11.86,+1.19,+0.11,-0.25,
     3.590,108.25, +9.3424202,57527.003,  0.2,  0.3, +2137.8, +651.6, 6,+0.00,+47.4129011,2.524,3.740,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     3.617,108.25, +9.3424202,57527.030,  0.2,  0.3, +2137.8, +651.6, 6,+0.00,+47.4129011,2.524,3.740,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     3.690,108.25, +9.3424202,57527.103,  0.2,  0.3, +2137.8, +651.6, 6,+0.00,+47.4129011,2.529,3.738,  0.000,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     3.691,108.25, +9.3424219,57527.400,  0.2,  0.3, +2137.8, +651.6, 6,+0.00,+47.4129028,2.529,3.738,  0.000,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     3.791,108.25, +9.3424219,57527.500,  0.2,  0.3, +2137.8, +651.6, 6,+0.00,+47.4129028,2.527,3.737,  0.033,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.12,-0.26,
     3.891,108.25, +9.3424219,57527.600,  0.2,  0.3, +2137.5, +651.5, 6,+0.00,+47.4129045,2.521,3.740,  0.066,  0.000,  0.000,  0.0,+0.00,11.86,+1.17,+0.12,-0.25,
     3.992,108.25, +9.3424219,57527.701,  0.2,  0.3, +2137.5, +651.5, 6,+0.00,+47.4129045,2.527,3.735,  0.098,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.25,
     4.093,108.25, +9.3424219,57527.802,  0.2,  0.3, +2137.5, +651.5, 6,+0.00,+47.4129045,2.527,3.744,  0.098,  0.000,  0.000,  0.0,+0.00,11.87,+1.18,+0.11,-0.25,
     4.193,108.25, +9.3424219,57527.902,  0.2,  0.3, +2137.5, +651.5, 6,+0.00,+47.4129045,2.526,3.741,  0.131,  0.000,  0.000,  0.0,+0.00,11.86,+1.18,+0.11,-0.26,
     4.293,108.25, +9.3424219,57528.002,  0.2,  0.3, +2137.1, +651.4, 6,+0.00,+47.4129061,2.527,3.735,  0.164,  0.000,  0.000,  0.0,+0.00,11.84,+1.18,+0.11,-0.26,
```

(Von der Empa zur Verfügung gestellt.)

Literaturverzeichnis

ACEM (18.10.2011). Motorcycles: EU registrations down 6% in first semester 2011. Gefunden am 18. Okt. 2011 unter `http://acem.eu/cms/det_pressreleases.php?det=1459`.

Bader, F. und Dorn, F. (1996). *Physik in einem Band*. Braunschweig: Schroedel.

Becker, A. und Pant, M. (2010). *Android 2. Grundlagen und Programmierung*. Heidelberg: dpunkt.

Deutsche Mathematiker-Vereinigung (18.10.2011). Sinus- und Cosinussatz. Gefunden am 18. Okt. 2011 unter `http://www.mathematik.de/ger/fragenantworten/erstehilfe/trigonometrie/sinuscosinussatz/sinuscosinussatz.html`.

Deutsches Zentrum für Luft- und Raumfahrt (DLR) (24.03.2011). Autofahren 2040: DLR simuliert Entwicklung des Fahrzeugmarktes. Gefunden am 18. Okt. 2011 unter `http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10081/151_read-689/year-2011/151_page-6/`.

Deutschschweizerische Mathematik- und Physikkommission (DMK und DPK) (1995). *Formeln und Tafeln*. Zürich: Orell Füssli.

Drews, R., Ahrens, B., und Bossmann, G.-E. (1991). *Messtechnik am Kraftfahrzeug, mobil und stationär*. Ehningen bei Böblingen: Expert Verlag.

eco-way (21.09.2011). Förderbeiträge für e-Bikes und e-Scooter. Gefunden am 18. Okt. 2011 unter `http://www.eco-way.ch/?p=2657`.

Fraunhofer Austria (18.10.2011). Trends in der Automobilindustrie. Gefunden am 18. Okt. 2011 unter `http://www.fraunhofer.at/publikationen/newsletter/report0111/trends_automobil.jsp`.

Google Inc. (19.10.2011f). Platform Versions. Gefunden am 19. Okt. 2011 unter `http://developer.android.com/resources/dashboard/platform-versions.html`.

Google Inc. (20.10.2011a). ADT Plugin for Eclipse. Gefunden am 20. Okt. 2011 unter `http://developer.android.com/sdk/eclipse-adt.html`.

Google Inc. (20.10.2011c). Google APIs Add-On. Gefunden am 20. Okt. 2011 unter `http://code.google.com/android/add-ons/google-apis/index.html`.

Google Inc. (20.10.2011d). Installing the SDK. Gefunden am 20. Okt. 2011 unter `http://developer.android.com/sdk/installing.html`.

Google Inc. (20.10.2011e). Open Source Project. Licenses. Gefunden am 20. Okt. 2011 unter `http://source.android.com/source/licenses.html`.

Google Inc. (20.10.2011h). What is the NDK? Gefunden am 20. Okt. 2011 unter `http://developer.android.com/sdk/ndk/overview.html`.

Google Inc. (21.10.2011b). android.database.sqlite. Gefunden am 21. Okt. 2011 unter `http://developer.android.com/reference/android/database/sqlite/package-summary.html`.

Google Inc. (22.10.2011g). What is Android? Gefunden am 22. Okt. 2011 unter `http://developer.android.com/guide/basics/what-is-android.html`.

Haken, K.-L. (2008). *Grundlagen der Kraftfahrzeugtechnik*. München: Carl Hanser.

heise online (20.10.2011). Galaxy Nexus: Google-Handy mit HD-Display und Android 4.0. Gefunden am 20. Okt. 2011 unter `http://heise.de/-1363207`.

Heissing, B., Ersoy, M., und Gies, S. (2011). *Fahrwerkhandbuch: Grundlagen, Fahrdynamik, Komponenten, Systeme, Mechatronik, Perspektiven*. Wiesbaden: Vieweg+Teubner.

Hucho, W.-H. (2008). *Aerodynamik des Automobils*. Wiesbaden: Vieweg+Teubner.

Mansfeld, W. (2010). *Satellitenortung und Navigation. Grundlagen, Wirkungsweise und Anwendung globaler Satellitennavigationssysteme*. Wiesbaden: Vieweg+Teubner.

Mitschke, M. und Wallentowitz, H. (2004). *Dynamik der Kraftfahrzeuge*. Berlin: Springer.

Parlamentsdienste (21.06.2011). CO_2-Reduktionsziele bleiben unverändert. Gefunden am 18. Okt. 2011 unter `http://www.parlament.ch/d/mm/2011/seiten/mm-urek-n-2011-06-21.aspx`.

Roberts, N., Andersen, D., Deal, R., Garet, M., und Shaffer, W. (1983). *Introduction To Computer Simulation: A System Dynamics Modeling Approach*. Reading, Massachusetts: Addison-Wesley.

Ruppelt, E. (2003). *Druckluft-Handbuch*. Essen: Vulkan-Verlag.

Schmidt, S. (2009). *Die Diffusion komplexer Produkte und Systeme. Ein systemdynamischer Ansatz*. Wiesbaden: Gabler.

Schöneborn, F. (2004). *Strategisches Controlling Mit System Dynamics*. Heidelberg: Physica-Verlag.

Simpligility Technologies Inc. (20.08.2010). Referential integrity with sqlite on Android the lazy way. Gefunden am 21. Okt. 2011 unter `http://www.simpligility.com/2010/08/referential-integrity-with-sqlite-on-android-the-lazy-way/`.

SQLite (21.10.2011a). Datatypes In SQLite Version 3. Gefunden am 21. Okt. 2011 unter `http://www.sqlite.org/datatype3.html`.

SQLite (21.10.2011b). SQLite Foreign Key Support. Gefunden am 21. Okt. 2011 unter `http://www.sqlite.org/foreignkeys.html`.

Stack Exchange Inc. (21.10.2011). Version of SQLite used in Android? Gefunden am 21. Okt. 2011 unter `http://stackoverflow.com/questions/2421189/version-of-sqlite-used-in-android`.

Stoffregen, J. (2010). *Motorradtechnik: Grundlagen und Konzepte von Motor, Antrieb und Fahrwerk*. Wiesbaden: Vieweg+Teubner.

System Dynamics Society (14.10.2011). Origin of System Dynamics. Gefunden am 14. Okt. 2011 unter `http://www.systemdynamics.org/DL-IntroSysDyn/origin.htm`.

The 4ViewSoft Company (21.10.2011). Achartengine. Gefunden am 21. Okt. 2011 unter `http://achartengine.org/`.

Ullenboom, C. (2008). *Java ist auch eine Insel*. Bonn: Galileo Press.

Vittore, C. (2006). *Motorcycle Dynamics*. Padova: Vittore Cossalter.

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungsverzeichnis