HOCHSCHULE harz
Hochschule für angewandte Wissenschaften (FH)

BOSCH
FV/SLH Hildesheim

Entwicklung einer videobasierten Fahrsimulation

Auf Basis realer Video- und GPS-Daten

Diplomarbeit

Patrick Kosiol

28. Oktober 2004

Prüfer HS-Harz:	Prof. Dr. Bernhard Zimmermann
Prüfer Robert Bosch GmbH:	Dipl.-Inf. Jens Fänger

Hochschule Harz
Fachbereich Automatisierung und Informatik

Thema und Aufgabenstellung der Diplomarbeit
DA AI 41/04

für Herrn Patrick Kosiol
geboren am: 06.07.80

Thema der Diplomarbeit:

Entwicklung einer videobasierten Fahrsimulation (auf Basis realer Video- und GPS-Daten)

Aufgabenstellung:

Autofahrern wird eine möglichst realitätsgetreue Fahrsimulation rechnergestützt zur Verfügung gestellt. Hierzu werden reale Autofahrten (Video & GPS) aufgenommen und in eine Form gebracht, die es dem Fahrer erlaubt, innerhalb dieser Daten zu navigieren. Diese Anwendung basiert auf JAVA sowie Video- und Kommunikations-bibliotheken unter Windows.

Ziel der Diplomarbeit ist es, durch den Einsatz von JAVA ein System zur Speicherung und Repräsentation raumbezogener Videodaten zu schaffen. Hierbei steht eine schnelle Implementierung und die Flexibilität der aufgenommenen Daten im Vordergrund. Die Bedienbarkeit soll möglist intuitiv und höchst funktional gestaltet werden. Außerdem soll das Programm Module enthalten, die es problemlos an bestehende Blaupunkt Navigationssysteme anbinden lässt.

Die Diplomarbeit beinhaltet folgende Teilaufgaben:

- Analyse der technischen Möglichkeiten eines derartigen Systems
- Konzeption der Fusion von raum- und videobezogenen Daten
- Prototypische Implementierungen
  - Erstellung eines Aufnahme-Moduls zur fusionierten Speicherung der Video- und GPS-Daten
  - Erstellung eines repräsentativen Fahrsimulators auf Basis der zuvor aufgezeichneten Daten
    - Anbindung dieses Simulators an bestehende Navigationssysteme
    - Steuerung der Fahrsimulation (z. B. durch erhöhen/verringern der Geschwindigkeit)
    - Umschaltmöglichkeit zwischen verschiedenen Fahrten einer Strecke (Tag/Nacht)

- Bewertung unterschiedlicher Lösungsansätze hinsichtlich der Realisierbarkeit und der Performance

Bearbeitungszeitraum: 30. 07. – 29. 10. 2004

Betreuer der Arbeit: Prof. Dr. Zimmermann

Dekan des Fachbereichs
Automatisierung und Informatik

i.V. T. Morgenstern

Prof. Dr. Zimmermann

Vorsitzender des Prüfungsausschusses des Fachbereichs
Automatisierung und Informatik

J. Krauser

Prof. Dr. Krauser

Danksagung

In erster Linie möchte ich mich bei meiner Familie bedanken, die mir stets sehr viel Unterstützung und Verständnis während meines Studiums entgegen brachte.

Des Weiteren möchte ich mich auch bei meinen beiden Betreuern Jens Fänger und Andreas Kynast der Firma Bosch bedanken, die mir bei der theoretischen und praktischen Durchführung meiner Diplomarbeit und der damit verbundenen Software mit Rat und Tat zur Seite standen.

Im Besonderen möchte ich aber auch Herrn Prof. Dr. B. Zimmermann danken, der mir während meines gesamten Hauptstudiums mit konstruktiver Kritik beistand und stets versuchte mir den richtigen Weg zu weisen.

Abstract (Deutsch)

Diese Diplomarbeit stellt Konzepte für die Entwicklung einer *videobasierten Fahrsimulation auf Basis realer Video- und GPS-Daten* vor. Eine solche Fahrsimulation findet in der HMI-Forschung der Abteilung FV/SLH der Firma Bosch seine Anwendung. Dabei besteht die Hauptproblematik in einer, nach Randbedingungen definierten, Synchronisation der Video- und GPS-Daten bei der Aufzeichnung und Wiedergabe dieser Daten. Somit werden hauptsächlich zwei große Themengebiete, die sich aus dieser Hauptproblematik ergeben, abgedeckt. Das erste Gebiet befaßt sich mit der synchronisierten Aufzeichnung von Videofahrten mit den dazu gehörigen GPS-Daten. Das zweite Gebiet behandelt die Präsentation dieser synchronisierten Daten mit Hilfe von multimedialen Werkzeugen, die interaktive Einflussnahme des Fahrers auf die Fahrsimulation sowie das Anbinden an bestehende Blaupunkt Navigationssysteme über standardisierte Protokolle zum Austausch von Ortungsdaten. Im Rahmen dieser Themengebiete werden unter anderem auch Konzepte zur Fusion mehrdimensionaler raum- und videobezogener Daten vorgestellt und bewertet.

Abstract (English)

This diploma thesis presents concepts for developing a *driving simulator based on real video- and GPS-data*. Such driving simulator will be applied in the field of HMI research by the department FV/SLH of the Bosch company. Thereby the main problem contains the synchronization of the video- and GPS-data during recording and playback periods under defined constraints. This main topic is covered by two parts within this work. The first part explains the synchronized recording of video- and the corresponding GPS-data. The second part discusses the presentation of synchronously recorded data using multimedia tools. Within the scope of this part are the introduction and evaluation of data fusion concepts for multidimensional space and video recordings. The second part also discusses ways of interactive influencing the simulation by the driver. Finally it covers connecting the simulation module to existing Blaupunkt navigation devices based on standardized protocols for positional data.

Inhaltsverzeichnis

Abbildungsverzeichnis X

Tabellenverzeichnis XII

Abkürzungsverzeichnis XIII

1 Einleitung 1

1.1 Human Machine Interaction . . . 1
1.2 Aufgabenstellung im Kontext von HMI . . . 1
1.3 Gliederung . . . 3

2 Software-Entwicklungskontext 4

2.1 Ergebnisorientierte Plattformwahl . . . 4
2.1.1 Überblick . . . 5
2.1.2 Multimedia . . . 6
2.1.3 Serielle Kommunikation . . . 7
2.1.4 Zusammenwirken von Multimedia und seriellen Daten . . . 7
2.2 Entwicklung von Software . . . 8
2.2.1 UML Modellierung . . . 8
2.2.2 Entwicklungsumgebung . . . 9
2.2.3 Quellcodeverwaltung . . . 9

3 Fusion von raum- und videobezogenen Daten 11

3.1 Problematik . . . 11
3.1.1 Videocharakteristika . . . 11
3.1.2 Ortungsdaten . . . 12
3.2 Lösungsansätze . . . 14
3.2.1 Einbetten von GPS-Daten in Einzelbilder . . . 14
3.2.2 Getrenntes Speichern von Video- und GPS-Daten . . . 16
3.3 Lösungskonzept der getrennten Speicherung von Video- und GPS-Daten . 18

4 Präsentation von raum- und videobezogenen Daten **21**
4.1 Problematik . . . 21
4.1.1 Eingangsparameter . . . 22
4.1.2 Ausgangsparameter . . . 22
4.2 Entwicklung eines Lösungskonzepts . . . 23
4.2.1 Videokomponente . . . 23
4.2.2 GPS-Komponente . . . 24
4.2.3 Zusammenwirken von Video und GPS . . . 27
4.3 Steuerung der Fahrsimulation . . . 29
4.3.1 Konzeptionelle Steuerungsmöglichkeiten . . . 29
4.3.2 Input für Geschwindigkeitssteuerungen . . . 30
4.4 Anbindung an bestehende Navigationssysteme . . . 31
4.4.1 Problematik und Zielsetzung . . . 31
4.4.2 Anbindung . . . 33

5 Fusion von mehrdimensionalen raum- und videobezogenen Daten **35**
5.1 Projekte, Strecken und Fahrten . . . 35
5.2 Problematik der räumlichen Fahrtenwechsel . . . 39
5.3 Lösungsansätze . . . 40
5.3.1 Vollständige Suche . . . 40
5.3.2 Lokale Suche . . . 41
5.3.3 Lokale Suche mit 'Keyframes' . . . 42
5.3.4 Indizierung aller GPS-Positionen . . . 43
5.4 Lösungskonzept Indizierung . . . 44
5.4.1 Mathematische Grundlagen . . . 46
5.4.2 Algorithmus zur Indexgenerierung . . . 50
5.4.3 Bewertung . . . 61

6 Fazit **62**
6.1 Fahrsimulation im Kontext von HMI . . . 62
6.2 Perspektiven . . . 64

Literaturverzeichnis **67**

A Spezifikationen **70**
A.1 Datenformat NMEA . . . 70
A.2 Datenformat FV [*vertraulich*] . . . 72
A.3 Dateiformat VSX . . . 72

B Das Java Media Framework **74**
B.1 JMF-Architektur und Datenmodelle 75
B.2 Darstellung von Videos in Java Components 78
B.2.1 Heavyweight- und Lightweight-Komponenten 78
B.2.2 AWT oder Swing für die Darstellung der Videos 79

C Steuerung von Multimedia Streams unter Java **81**
C.1 Anbinden von DirectInput-Geräten über das Native Interface von Java . . 81
C.2 Keyboardsteuerung der Fahrsimulation 83
C.3 Proportionalitäten im Bereich der CPU-Auslastung 83

D Daten **85**
D.1 Aufgenommene Strecken und Routen 85

Abbildungsverzeichnis

2.1 Prozesskette des Aufnahme- und Simulationsmoduls 5
2.2 JMF Architektur (Quelle: [JMF-Guide]) 7
2.3 Systemüberblick 8

3.1 Kodierung eines Videoeinzelbildes 15
3.2 In ein Bild eingebettete GPS-Daten 15
3.3 Synchronisation von Video und GPS 17

4.1 Anforderungsvisualisierung 21
4.2 Verarbeitung der Videodaten 23
4.3 Verarbeitung der GPS-Daten 25
4.4 GPS-, Gyro- und GALA-Datenintervall 26
4.5 GPS-Datenmodellierung 26
4.6 GPS/Video Synchronisation 28
4.7 Achsenaufteilung „Microsoft SideWinder ForceFeedback Wheel“ 31
4.8 GPSWriter-Architektur 33
4.9 Anbindungsstruktur von externen Geräten 34

5.1 *Project-Track-Run* UML-Klassendiagramm I 36
5.2 *Project-Track-Run* Architektur - Aufbau 37
5.3 *Project-Track-Run* Architektur - Anwendung 37
5.4 *Project-Track-Run* UML-Klassendiagramm II 38
5.5 *GPSContainer* Architektur 38
5.6 Umschaltung zwischen verschiedenen Fahrten einer Strecke 39
5.7 Vollständige Suche 41
5.8 Lokale Suche 42
5.9 Beispielhafte Einbettung von Keyframes 43
5.10 Indizierung von 2 Fahrten 44
5.11 Fusionsrasterung von 2 Fahrten 46
5.12 Aneinandergereihte GPS-Koordinaten 47
5.13 Vergrößert dargestellte Rasterung auf Basis von Abbildung 5.12 47

5.14 Senkrechte Vektorprojektion (Quelle: [StrickWurl, Seite 52]) 49
5.15 Beispiel - Drei Fahrten einer Strecke 54
5.16 Beispiel - Drei Fahrten einer Strecke - Zwischenvektoren 55
5.17 Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 1 55
5.18 Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 1 *Index* 56
5.19 Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 2 57
5.20 Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 2 *Index* 58
5.21 Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 3 58
5.22 Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 3 *Index* 59
5.23 Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 4 59
5.24 Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 4 *Index* 60

6.1 Screenshot *VideoSim* 63
6.2 *VideoSim* im Fahrsimulator zur HMI-Forschung 64

A.1 VSX Beispieldatei 73

B.1 JMF *DataSource* Architektur (Quelle: [JMF-Guide]) 75
B.2 UML-Klassendiagramm zur Verwaltung von JMF-Video-Daten in *VideoSim* 76
B.3 JMF *Player*/*Processor* Architektur (Quelle: [JMF-Guide]) 76
B.4 Zustandsdiagramm *Player* (Quelle: [JMF-Guide]) 77
B.5 Zustandsdiagramm *Processor* (Quelle: [JMF-Guide]) 77
B.6 Verarbeitung von Multimediadaten in einem *Processor*(Quelle: [JMF-Guide]) 78

C.1 JNI-DirectInput-Achitektur 82
C.2 Microsoft SideWinder ForceFeedback Wheel (Quelle: [MS]) 82
C.3 BMW-Multifunktionslenkrad der Baureihe E38 83
C.4 Proportionalitäten im Bereich der CPU-Auslastung 84

D.1 Logische Streckenübersicht 85
D.2 Graphische Streckenübersicht 86
D.3 Graphische Streckenübersicht - Strecke 1 87
D.4 Graphische Streckenübersicht - Strecke 2 88
D.5 Graphische Streckenübersicht - Strecke 3 89
D.6 Graphische Streckenübersicht - Strecke 4 90
D.7 Graphische Streckenübersicht - Strecke 5 91
D.8 Graphische Streckenübersicht - Strecke 6 92
D.9 Graphische Streckenübersicht - Strecke 7 93
D.10 Graphische Streckenübersicht - Strecke 8 94
D.11 Graphische Streckenübersicht - Strecke 9 95

Tabellenverzeichnis

3.1 Beispiele - NMEA Protokollauszüge 13
3.2 Beispiele - FV Protokollauszüge 13
3.3 Beispiele - Video-Zeitstempel mit zugehörigen FV-Frames 19

4.1 Videoabspiel- und Bildraten im Verhältnis 29
4.2 Achsenaufteilung „Microsoft SideWinder ForceFeedback Wheel“ 31

5.1 Definitionen für das mathematische Prinzip der senkrechten Vektorprojektion 48
5.2 Definitionen für den Pseudo-Code der Indizierung 52

A.1 NMEA RS232 Konfiguration (Quelle: [NMEA-Spec2]) 70
A.2 NMEA Spezifikationsbeispiel *$GPGGA* 71
A.3 NMEA Spezifikationsbeispiel *$GPVTG* 72

C.1 Legende zu Abbildung C.2 82

D.1 Streckenauflistung aus Abbildung D.1 und D.2 86

Abkürzungsverzeichnis

API	Application Programming Interface
AWT	Abstract Windowing Toolkit
CoDec	Compressor Decompressor
COM-Port	Communication Port
CVS	Concurrent Version System
DV	Digital Video
DVSD	Digital Video Standard Definition
Gala	Geschwindigkeitsabhängige Lautstärkenanpassung
GPS	Global Positioning System
GUI	Graphical User Interface
HMI	Human Machine Interaction
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineers
JMF	Java Media Framework
MIDI	Musical Instrument Digital Interface
MiniDV	Mini Digital Video
MPEG	Moving Picture Expert Group
NMEA	National Marine Electronics Association
ODM	Ortungsdatenmodul
OSGi	Open Services Gateway Initiative

PDV	Panasonic Digital Video
RS232	Recommended Standard 232
RTP	Real Time Transport Protocol
RTSP	Real Time Streaming Protocol
RTCP	Real Time Control Protocol
SDK	Software Development Kit
TCP/IP	Transmission Control Protocol / Internet Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTC	Universal Time Coordinated
VSX	VideoSim XML
WGS84	World Geodetic System 1984
XML	Extensible Markup Language

1 Einleitung

Im Folgenden wird das Umfeld der Diplomarbeit sowie deren Randbedingungen dargestellt. Anschließend wird die Aufgabenstellung innerhalb dieses Umfelds eingeordnet und deren Motivation im Zusammenhang betrachtet.

1.1 Human Machine Interaction

Der Bereich Human Machine Interaction oder auch Mensch-Maschine-Schnittstelle der Abteilung FV/SLH der Robert Bosch GmbH befasst sich mit intelligenten Bedienassistenzkonzepten im Fahrzeug.

„[...] Der Fahrer hat während der Fahrt nur wenig Zeit, sich der Bedienung von Verkehrstelematik-, Navigations- und Infotainmentsystemen zu widmen. Hier sind neue Bedienkonzepte gefordert, die dem Fahrer 'assistieren', sich an seine Vorlieben anpassen, die Fahrsituation berücksichtigen, Informationen und Angebote prüfen und neu strukturieren sowie Eingabemasken der Diensteanbieter auf ein einheitliches Bedienkonzept abbilden. Daraus folgt eine intelligente, adaptive Bedienassistenz (User Centered Assistance). Ziel dieser HMI Forschung bei Bosch ist unter anderem die Entlastung des Fahrers durch eine Verringerung der Entscheidungsebenen. [...]“ (Quelle: [Kynast])

1.2 Aufgabenstellung im Kontext von HMI

Im Rahmen von Versuchsreihen soll untersucht werden, in wie weit Verkehrstelematik-, Navigations- und Infotainmentsysteme den Fahrer von seiner eigentlichen Aufgabe, dem Führen des Fahrzeugs, ablenken. Dabei gibt es zwei grundsätzliche Ansätze, die eine derartige Untersuchung ermöglichen können:

1. Fahren in einem PKW
2. Fahren in einer Fahrsimulation

Ersteres bietet den Vorteil der schnellen Verfügbarkeit, da keine Software entwickelt werden muss, und einer absolut realitätsgetreuen Versuchsdurchführung. Ein daraus entstehender Nachteil ist die fehlende Reproduzierbarkeit des Versuchs. Die Erforschung der

Reaktion verschiedener Probanden auf absolut identische Vorkommnisse ist ein wichtiger Aspekt in diesem Gebiet der HMI Forschung. Um dies zu gewährleisten besteht die Möglichkeit der Modellierung einer Fahrsimulation, die genau diese Versuchsdurchführungen reproduzierbar macht. Im Bereich der Fahrsimulationen gibt es drei grundlegende Ansätze:

1. Freies Fahren in 3D-Arealen
2. Fahren auf vorgegebenen generierten Strecken auf Basis eines Streckennetzes
3. Fahren in zuvor aufgezeichneten Videofahrten

Es wurde auf Basis des bundesdeutschen Streckennetzes eine Fahrsimulation entwickelt, die aus gegebenen Straßendaten eine 3D-Landschaft in Echtzeit generiert, in der sich der Fahrer frei bewegen kann. Dies wäre im Bereich einer Mischform aus 1. und 2. einzuordnen, da sich der Fahrer zum Teil frei in der generierten 3D-Landschaft bewegen kann und dennoch ein zugrunde liegendes Streckennetz zur Verfügung hat. Derartige Modellierungen der Realität werden jedoch nicht ohne einem hohen Maß an Aufwand und Entwicklungskosten realitätsgetreue Landschaften generieren können. Die zu Grunde liegende digitale Straßenkarte beinhaltet ausschließlich Informationen über die Straße selbst, also beispielsweise Fahrbahntyp, Anzahl der Fahrspuren, zugelassene Höchstgeschwindigkeit und ähnliche Attribute. Besonderheiten neben der Strecke, wie markante Häuser, Bäume oder auch eindrucksvolle Landschaftskonstellationen sind darin nicht abgebildet und können somit auch nicht generiert werden. Die Vorteile einer *Fahrsimulation basierend auf realen Strecken- (GPS) und Bilddaten (Video)* sind:

1. Dem Fahrer das Gefühl zu geben, er bewege sich im realen Straßenverkehr. Um dies zu bewerkstelligen, wird ein Fahrzeug mit einer Kamera und einem GPS-Empfänger ausgestattet. Bei einer Fahrt entlang einer Referenzstrecke werden die GPS- und Videodaten synchron aufgezeichnet.
2. Es werden alle möglichen Störstellen während einer Autofahrt (die Fahrbahn kreuzende Busse, impulsive Fußgänger, heftiges Abbremsen wie auch verschiedene Wettersituationen) aufgezeichnet und können somit in einer Fahrsimulation auch wieder dargestellt werden.
3. Aus Punkt 3 ergibt sich weiterhin die Möglichkeit, Probanden mit verschiedenen Fahrsituationenen zu konfrontieren, die seine Aufnahmefähigkeit überlasten könnten. Da diese verschiedenen Fahrsituationen aufgezeichnet sind, entsteht eine hundertprozentige Reproduzierbarkeit bei Versuchen mit unterschiedlichen Probanden.
4. Verschiedene Wetter- und Verkehrslagen bieten unterschiedliche Belastungsgrade.

Diese Vorteile bilden die Motivation für die *Entwicklung einer videobasierten Fahrsimulation auf Basis realer Video- und GPS-Daten*. Aufgrund der Erkenntnisse, die sich durch Versuchsreihen mit einer solchen Fahrsimulation ergeben, ist es möglich, bestehende Navigationssoftware zu optimieren und neue Arten der Zielführung (z.B. über Orientierungspunkte) zu entwickeln.

1.3 Gliederung

Diese Diplomarbeit unterteilt sich in drei Bereiche. Die ersten beiden Kapitel, Einleitung und Software-Entwicklungskontext, führen an die Thematik heran. Sie dienen dem Verständnis der Aufgabenstellung in einem bestimmten Kontext sowie der Klärung von Architekturfragen im Bezug auf die Aufgabenstellung und der sich daraus ergebenden Randbedingungen. Daraufhin folgen die Kapitel 3, 4 und 5, welche den Haupteil und somit den inhaltlichen Schwerpunkt der Arbeit darstellen. Hierbei wird von den letztendlichen Implementierungen abstrahiert. Der Inhalt dieser drei Hauptteile ist stets ähnlich aufgebaut. Zu Beginn wird die Problemstellung beschrieben, danach werden verschiedene Lösungsansätze vorgestellt, aus denen sich letztendlich ein Lösungskonzept ergibt. Dieses soll jedoch nicht als endgültig im Raum stehen, sondern eher zur Diskussion anregen. Die favorisierten Lösungsansätze sind in die Entwicklung der Fahrsimualtion, welche im Rahmen der Erstellung dieser Diplomarbeit stattfand, eingeflossen. Am Ende des Hauptteils wird eine Zusammenfassung und mögliche Perspektiven, aufbauend auf den dargestellten Konzepten, offeriert.

2 Software-Entwicklungskontext

2.1 Ergebnisorientierte Plattformwahl

Ziel der Entwicklung soll eine schnell zu implementierende prototypische Software mit hoher Funktionalität sein. Dabei steht jedoch nicht die Entwicklung der Software zu einem 'verkaufsfähigen' Produkt im Vordergrund. Viel mehr soll mit dieser Fahrsimulation ein Prototyp entstehen, anhand dessen Untersuchungen durchgeführt werden können.

Zur Wahl stehen in diesem Bereich zwei essentielle Entwicklungsplattformen: Java und C++. In Bezug auf die zuvor genannte Zielstellung bietet Java einige Vorteile. Vor allem die schnellen Implementierungsmöglichkeiten, frei verfügbare Dokumentationen und Bibliotheken sowie eine große Programmiergemeinde machen Java hier interessant. Zur Aufnahme und Verarbeitung von Multimediadaten steht das Java Media Framework von Sun Microsystems zur Verfügung. Des Weiteren muss ein Ortungsdatenmodul via COM- oder USB-Port angesteuert werden und GPS-Daten liefern können. Hierfür stellt Sun Microsystems wiederum ein auf Java basierendes Kommunikationspaket names 'javax.comm' zur Verfügung, welches kompfortable Zugriffsbibliotheken anbietet. Das Zusammenspiel dieser einzelnen Elemente (JMF & javax.comm) lässt sich demnach hervorragend in bestehende Systeme integrieren. Die Kombination aus diesen beiden Komponenten ist nicht die einzige Möglichkeit eine solche Problemstellung zu lösen, bietet jedoch Vorteile in der Implementierung. Sie erlaubt ein schnelles und komfortables Arbeiten, was gerade mit C++ und entsprechenden Tools (soweit vorhanden) nicht immer gegeben ist aber dennoch möglich wäre. Deshalb wurden diese Komponenten (Java, JMF & javax.comm) für die Entwicklung von Software, welche im Rahmen der Erstellung dieser Diplomarbeit erfolgte, eingesetzt.

2.1.1 Überblick

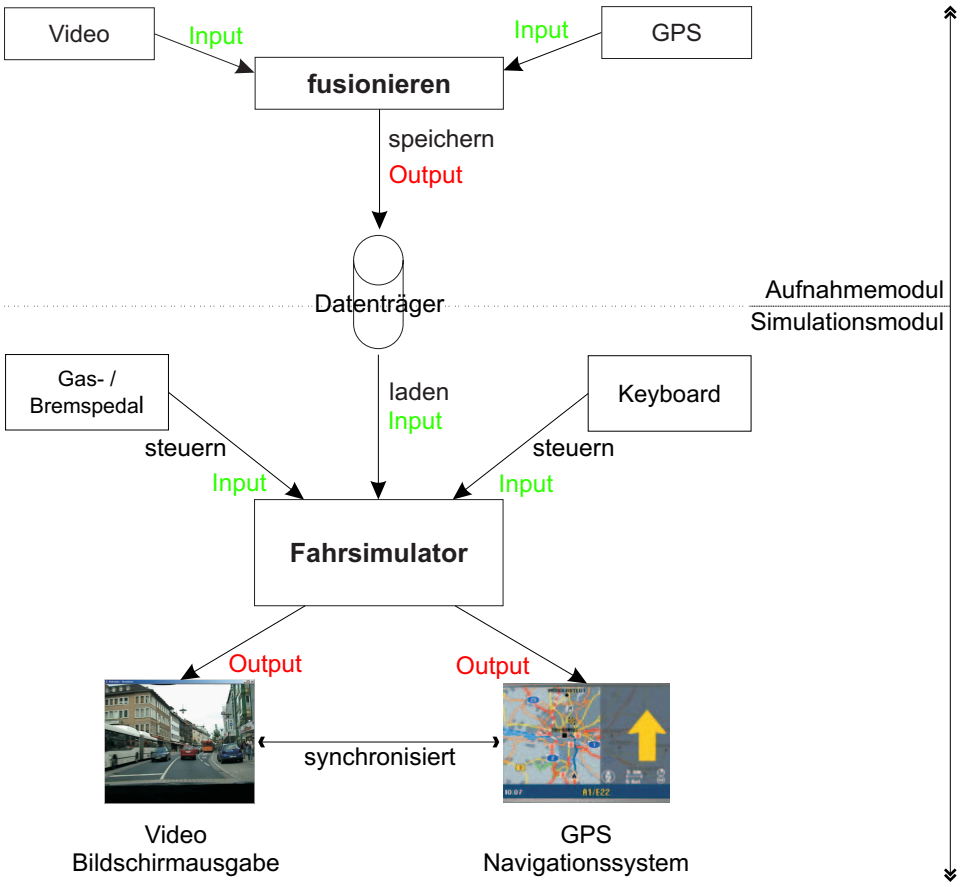


Abbildung 2.1: Prozesskette des Aufnahme- und Simulationsmoduls

Die Abbildung 2.1 zeigt die komplette Prozesskette der im Rahmen der Diplomarbeit erstellten Softwaremodule. Der obere Bereich, in dem das Aufnahmemodul dargestellt ist, zeigt die beiden Eingangsparameter Video und GPS. Diese werden in einer Softwarekomponente zusammengeführt und danach auf einem Datenträger gespeichert. Unterhalb des Datenträgers befindet sich das Simulationsmodul, welches zur Präsentation, Steuerung und Weiterverarbeitung der zuvor aufgenommenen Daten dient. Dabei sind die Eingangsgrößen dieses Fahrsimulators die Daten auf dem Datenträger (Schnittstelle zwischen Aufnahme- und Simulationsmodul), ein Gas- bzw. Bremspedal sowie das Keyboard. Ebenso liefert der Fahrsimulator auch Ausgangsgrößen in Form der Präsentation des aufgenommenen Videos auf dem Bildschirm und der Bereitstellung der synchronisierten GPS-Daten für angebundene Navigationssysteme.

Im Folgenden soll nun geklärt werden, inwieweit diese Komponenten mit den jeweiligen Hardwaregeräten zusammenarbeiten und wie die Kombination daraus zu einem System zusammenwachsen kann.

2.1.2 Multimedia

Um Videodaten aufnehmen und visualisieren zu können, werden weitreichende Multimediabibliotheken benötigt. Aufgrund der getroffenen Plattformwahl gibt es bestimmte Bewertungskriterien, an denen sich diese zusätzlichen Komponenten orientieren müssen.

Da bei einer MiniDV-Kamera die Daten bereits digital vorliegen, müssen diese nicht in Echtzeit digitalisiert und komprimiert werden. Prinzipiell können sogar mehrere Videos parallel aufgezeichnet werden. Eine MiniDV-Kamera[1] arbeitet auf Basis des MPEG2-Videoformats, welches im DVSD[2]-Format enthalten ist. Derartige Geräte werden über USB oder FireWire[3] an Datenverarbeitungseinheiten, wie den Computer, angeschlossen und können somit Daten austauschen. In erster Linie handelt es sich hierbei um Videodaten bzw. Videostreams, die exportiert werden, da das Hauptaugenmerk auf einer *videobasierten* Fahrsimulation liegt. Audiodaten können in diesem Fall ignoriert werden, da diese nur von dem Innenraum des Autos stammen würden und nicht simulationsrelevant sind.

Das Java Media Framework ist ein Paket von Bibliotheken zur Implementierung von Multimediaanwendungen unter Java. Mit ihm lassen sich Audio- und Videoanwendungen verschiedenster Art entwickeln. Das JMF bietet Funktionen zur Präsentation und Verarbeitung solcher Daten. Außerdem setzt auf Basis dieser JMF-API eine weitere JMF-Plug-In-API auf, welche die Integration von Plug-In-Modulen erlaubt. Hiermit ist es möglich, eine Vielzahl unterschiedlicher Funktionalitäten in das bestehende JMF-System einzubinden. Wenn beispielsweise die Standardimplementation der VideoRenderer nicht so performant ist wie gewünscht, dann besteht die Möglichkeit ein eigenes derartiges Plug-In zu integrieren. Darauf aufbauend lässt sich die „High-Level-Architektur des Java Media Frameworks“[4], wie in Abbildung 2.2 gezeigt, illustrieren.

[1] Hier eine Sony DCR-PC109E.
[2] DVSD bezeichnet die „Digital Video Standard Definition“ und ist ein digitales Videoformat, welches FireWire- und Digital-Video-basierte Aufnahmegeräte benutzen.
[3] Multimediale Kommunikationsschnittstelle standardisiert nach IEEE 1394.
[4] Quelle: [JMF-Guide].

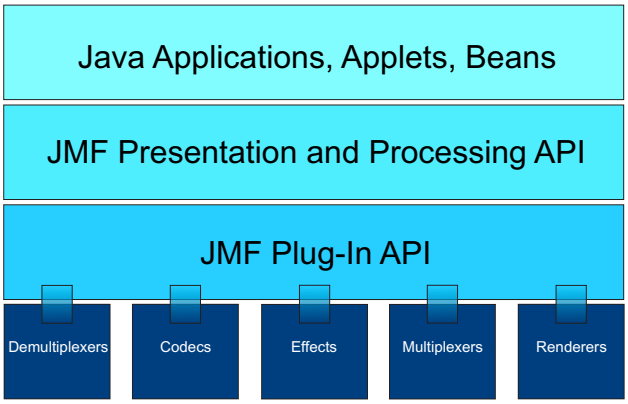


Abbildung 2.2: JMF Architektur (Quelle: [JMF-Guide])

2.1.3 Serielle Kommunikation

Um weitergreifende Informationen zur Erstellung einer, der Aufgabenstellung entsprechenden Fahrsimulation zu erhalten, muss die Möglichkeit bestehen, originale GPS-Datensätze zu den jeweiligen Videos zu empfangen, aufzuzeichnen und wieder zu senden. Hierzu stellt die Robert Bosch GmbH ein Ortungsdatenmodul zur Verfügung. Es liefert Ortungsdaten über eine RS232 Schnittstelle (COM-Port) an verschiedenste Geräte und arbeitet somit unabhängig vom angeschlossenen Endgerät.

Die, von Sun Microsystems frei zur Verfügung gestellte und favorisierte, Java Bibliothekensammlung 'javax.comm' stellt stabile Zugriffsmöglichkeiten für die Kommuniktaion mit seriellen und parallelen Schnittstellen bereit. Somit ist die Kommunikation zwischen einer Java Applikation und dem Ortungsdatenmodul sichergestellt.

2.1.4 Zusammenwirken von Multimedia und seriellen Daten

Aus den beiden vorhergehenden Abschnitten (2.1.2 & 2.1.3) geht hervor, dass die Komponenten JMF und javax.comm in einer Plattform aufgehen müssen. Da beide Komponenten auf der Java Technologie basieren, arbeiten sie mit dem Java SDK[5] zusammen. Das JMF setzt, wie auch das javax.comm Paket, auf der Basis des Java SDKs von Sun Microsystems auf und kann somit in die verschiedensten Java Applikationen integriert werden.

Abbildung 2.3 zeigt die prinzipielle Zusammenarbeit der drei Hauptkomponenten Java SDK, JMF, jaxax.comm mit dem Betriebssystem und deren Schnittstellen zu Hardwaregeräten, wie einer Digitalkamera und einem GPS-Empfänger. Wie in der unteren Abbildung zu erkennen ist, wird eine MiniDV-Kamera per FireWire (IEEE 1394) angebunden. Der Videostrom der MiniDV-Kamera wird über diese Schnittstelle gespeist und in der JMF-Java-Umgebung verarbeitet. Zeitgleich geschieht dies auch mit einem Ortungsdatenmodul

[5] Java Software Development Kit der Firma Sun Microsystems.

(ODM). Dieses ist über eine serielle Schnittstelle (COM- bzw. USB-Port) angebunden und sendet wiederum kontinuierlich Ortungsdaten an das System. Die Verarbeitung erfolgt parallel zu den Videodaten.

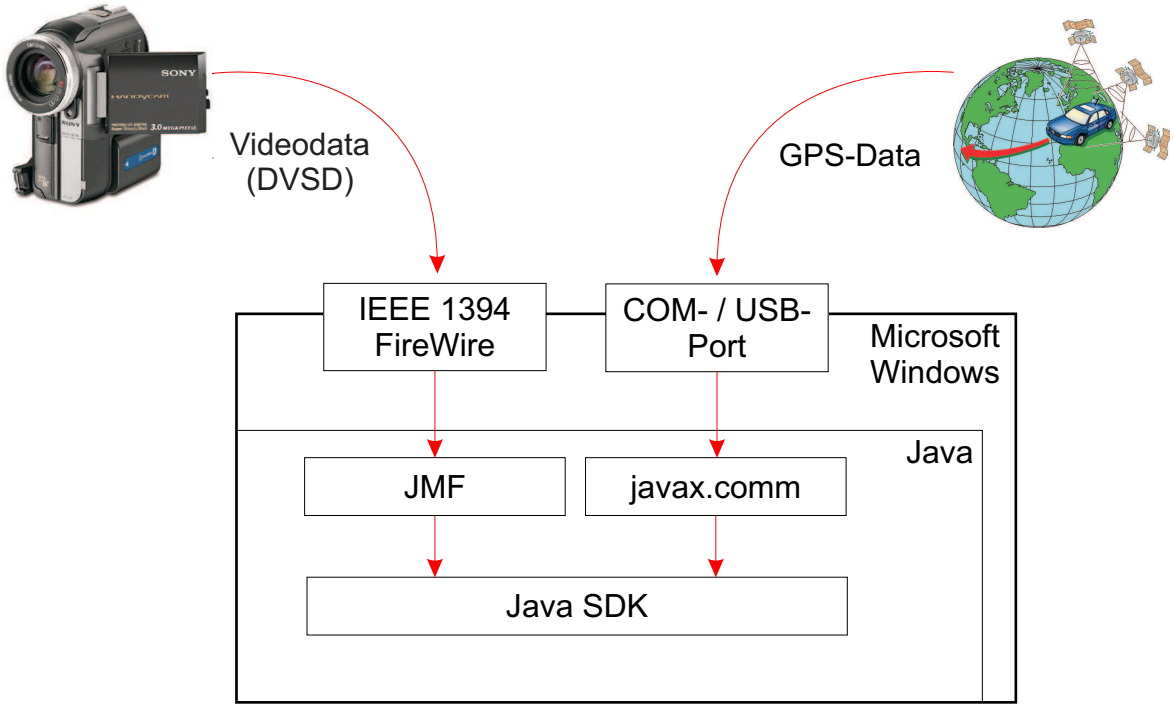


Abbildung 2.3: Systemüberblick

2.2 Entwicklung von Software

2.2.1 UML Modellierung

Der aktuelle Stand der allgemeinen Softwareentwicklung erlaubt es, sich auf Abstraktionsebenen zu bewegen, die losgelöst von den zu verwendenden Werkzeugen sind. In diesem Gebiet spielt die Modellierung zu entwickelnder Systeme eine tragende Rolle. Hierbei ist die Unified Modeling Language[6] zu erwähnen. Diese Sprache kann für die Visualisierung, Spezifikation, Konstruktion und Dokumentation von Softwaresystemen verwendet werden. Hierbei handelt es sich um graphische Darstellungen von Programmen, Erstellung von präzisen, eindeutigen und vollständigen Modellen sowie deren Architekturabbildungen. Aus diesen UML-Modellen kann dann wiederum Programmcode in verschiedenen

[6] Quelle: [UML].

Sprachen generiert werden. Auf dem weltweiten Markt für derartige Software gibt es zwei bedeutende Systeme, Rational Rose von IBM[7] und Together von Borland[8].

Im Rahmen dieses Projekts wird die Software „Together“ eingesetzt, um aus modellierten UML-Diagrammen lauffähigen Java-Programmcode zu erzeugen. Auf der Basis dieser Systementwicklungen und -generierungen entsteht ein Fundament, eine Klassenstruktur, die um die gewünschten Funktionalitäten erweitert werden kann. Dieses Fundament trägt zur Entwicklung eines Endprodukts einen großen Anteil bei.

2.2.2 Entwicklungsumgebung

Eine sorgfältig ausgewählte IDE versetzt den Softwareentwickler in die Lage, ein aus UML-Diagrammen erzeugtes Grundgerüst um die notwendigen Funktionalitäten zu erweitern. Dies kann wiederum mit Together geschehen, da es eine IDE-ähnliche Oberfläche integriert hat. Der Markt für IDEs in diesem Bereich ist ziemlich stark fragmentiert. Zu den großen Playern zählen hier unter anderem Microsoft, Borland, Sun Microsystems und IBM, wobei die OpenSource IDE Eclipse aufgrund seiner Plug-In-Architektur und der hohen Funktionalität sehr vielseitig ist. Sie wird in einem Konsortium unter der Federführung von IBM und Sun Microsystems frei zur Verfügung gestellt und stetig weiter entwickelt. Eclipse setzt auf dem Prinzip des OSGi[9] auf und bietet hervorragende Funktionen und Erweiterungsmöglichkeiten für Softwareentwickler. Außerdem harmoniert es sehr gut mit externen Programmen wie beispielsweise Together, Rational Rose oder auch CVS.

2.2.3 Quellcodeverwaltung[10]

Das Concurrent Versions System, kurz CVS genannt, ist ein OpenSource Werkzeug, welches ein netzwerktransparentes Versionskontrollsystem zur Verfügung stellt. CVS ist hilfreich in den verschiedensten Gebieten, vom einzelnen Entwickler bis hin zu verteilten Entwicklungsteams. Es besitzt folgende Merkmale:

- Client-Server-Zugriffsmechanismen geben den Entwicklern die Möglichkeit von überall aus über das Internet den aktuellsten Quellcode herunterzuladen.
- Das Check-Out-Modell für die Versionskontrolle vermeidet vorhersehbare Konflikte.

[7] bzw. deren mittlerweile 100%igen Tochter Rational

[8] bzw. deren mittlerweile 100%igen Tochter TogetherSoft

[9] „[...] Die Spezifikation der OSGi Service Platform definiert eine Java-basierte Laufzeitumgebung und Basisdienste. Ein bedeutendes Merkmal der Service Platform ist die Möglichkeit, dynamisch und kontrolliert Service-Anwendungen (sog. Bundles) zur Laufzeit einspielen und - vor allem - auch wieder entfernen zu können. Das Modell der OSGi Service Platform gibt damit die Möglichkeit, verschiedene, weitgehend unabhängige Anwendungen in der selben Java Virtual Machine (VM) laufen zu lassen. Ein Einsatz von OSGi ist die Integrierte Entwicklungsumgebung Eclipse. Plugins für Eclipse sind (seit Version 3) OSGi Bundles. [...]“ (Quelle: [Wiki-OSGi])

[10] Quelle: [CVSHOME].

- Die Client-Anwendungen sind für die meisten Plattformen verfügbar.

Somit bietet CVS eine ideale Möglichkeit, um die Entwicklung von Software zu überwachen, zu dokumentieren und, bei der verteilten Entwicklung mit mehreren Personen, die Synchronisation des Quellcodes zu gewährleisten.

3 Fusion von raum- und videobezogenen Daten

3.1 Problematik

Das Ziel der Diplomarbeit ist eine *Fahrsimulation basierend auf realen Video- und GPS-Daten*. Dies setzt voraus, dass es ein intelligentes System zur Erstellung bzw. Aufzeichnung derartiger Daten gibt. Eine solche Fahrsimulation muss demzufolge aus zwei elementaren Komponenten bestehen. Einem Modul zur Speicherung von Videodaten sowie einem Pendant für GPS-Daten. Das simple Abspeichern dieser beiden Datensätze reicht hierbei nicht aus. Daraus folgend ist eine sichere Synchronisation der beiden Sensorquellen (Video & GPS) erforderlich.

3.1.1 Videocharakteristika

Ein Video kann als ein Fluss von Daten betrachtet werden. Ein maßgeblicher Ansatzpunkt wissenschaftlicher Betrachtungen von Videodaten ist die Performance. Ein System, welches Videos nur unter vollständiger Auslastung der CPU verarbeiten kann, verliert mit hoher Wahrscheinlichkeit wichtige Daten und verringert somit die Qualität des letztendlich aufgenommenen Videos.

Prinzipiell ist ein Video eine Aneinanderreihung von Einzelbildern (Frames), welche mit einer bestimmten Frequenz (Framerate) abgespielt wird. Diese Framerate liegt für handelsübliche TV-Geräte und DV-Kameras bei 25 Bildern pro Sekunde. In diesem Bereich vermag das menschliche Auge die einzelnen Bilder nicht mehr von einander zu trennen und die einzelnen 'Schnappschüsse' werden als bewegte Bilder wahrgenommen. Bei einer schlechten, sprich langsamen, Verarbeitung von Videodaten verringert sich demzufolge die Framerate. Ab einem gewissen Schwellwert fängt das menschliche Auge an, wieder die Einzelbilder wahr zu nehmen und es geht die Illusion eines fortlaufenden Videos verloren. Bei einem unregelmäßigen Verlust von einzelnen Videoframes (dem ständigen Variieren der Framerate) kann gar der Bewegungsfluss des Videos komplett verloren gehen. Die Bewertung videoverarbeitender Module muss an folgenden Gesichtspunkten orientiert werden:

- Geschwindigkeit

- Qualität des Videos[1]
- Realisierbarkeit / Integrierbarkeit

3.1.2 Ortungsdaten

Um die Position des Fahrzeugs bestimmen zu können, wird das Global Positioning System (GPS) eingesetzt, welches derzeit als quasi Monopolist einen Standard definiert. Diese GPS-Daten müssen dabei so verarbeitet und gespeichert werden, sodass bei der Verwendung der Fahrsimulation ein Ortungsdatenmodul simuliert werden kann. Somit besteht die Möglichkeit aus dieser Fahrsimulation heraus Navigationsgeräte mit GPS-Signalen zu speisen.

GPS allgemein

Das Global Positioning System bezeichnet die eindeutige globale Positionsbestimmung eines Objekts auf der Erde. Dieses, von der U.S. Army betriebene, System verfügt über eine Vielzahl von Ortungssatelliten im geostationären Orbit. Ein, mit einem GPS-Modul ausgestattetes, Fahrzeug auf der Erde kann mit Hilfe dieses Systems seine Position, Fahrtrichtung, Geschwindigkeit, Höhe über Normal Null und weitere Informationen bestimmen. Bei entsprechender Genauigkeit kann dieses Fahrzeug somit präzise zu jeglichen gewünschten Zielen in dem weltweiten Straßennetz[2] geführt werden.

GPS-Datenformate

Die Basisaufgabe von Modulen zum Empfang von GPS-Daten ist es, diese an anderweitige Geräte wie beispielsweise Navigationssysteme oder auch PCs über definierte Schnittstellen zu liefern. Hierzu werden Protokolle benötigt, die diesen Datenaustausch bewerkstelligen und standardisieren, um Interoperabilität zu erreichen. Aufgrund der Herkunft des GPS aus der Schifffahrt hat die NMEA (National Marine Electronics Association) unter anderem den Standard NMEA-0183 (in stetig aktualisierten Versionen) verabschiedet. Hierbei werden die Daten, beispielsweise über den COM-Port[3], im Textformat zur Verfügung gestellt. Dieser Standard sieht vor, GPS-Datensätze nach vordefinierten Inhalten zu trennen. Dabei hat ein jeder Datensatz einen Präfix, der den Typ definiert, anhand dessen erkannt werden kann, welche Daten in welcher Reihenfolge in diesem String enthalten sind. Im Folgenden sind drei NMEA-Beispielstrings[4] aufgeführt:

[1] Das Videobild muss möglichst scharf und gut sichtbar dargestellt werden können.
[2] Vorausgesetzt es wurde kartographiert.
[3] Die serielle Schnittstelle eines Computers.
[4] Detaillierte Beschreibungen dieser NMEA-Beispiele befinden sich im Anhang A.1 auf Seite 70.

Bedeutung	Beispiel
GPS-Positionen	$GPGGA,170111,4338.581,N,07015.101,W,1,00,2.0,1.1,M,-31.8,M,,*71
Geschwindigkeit/Richtung	$GPVTG,054.7,T,034.4,M,005.5,N,010.2,K*4A
Zeit/Datum	$GPZDA,201530.00,04,07,2002,00,00*6E

Tabelle 3.1: Beispiele - NMEA Protokollauszüge

Des Weiteren wird von der Abteilung FV/SLH der Robert Bosch GmbH ein eigenes Ortungsdatenmodul (ODM) für den Einsatz im Fahrzeug entwickelt. Es hat die Möglichkeit, die angesprochenen GPS-Daten nicht nur im NMEA-Format, sondern auch in dem Bosch-eigenen FV-Format zu liefern. Dieses Format basiert nicht auf Strings (vgl. NMEA), sondern verschickt die Daten in Rohform als Byteframes. Aufgrund seines Aufbaus ist das ODM dadurch in der Lage, die angesprochenen Frames mit einer höheren Frequenz bereit zu stellen. Aus den folgenden Beispielframes[5] können die GPS-Informationen extrahiert und weiter verarbeitet werden.

Bedeutung	Beispiel
GPS-Positionen inkl. GALA/Gyro	51 16 123 15 -1 17 5 7 -44 2 -26 -77 -96 0 79 120 89 0 14 -107 108 0 0 3 52 0 0 0 13 0 0 19 -20 47 1 4 2 -76 0 0 0 -55 0 0 0 -65 -57 -76 6 10 -66 -105 0 6 38
nur GALA/Gyro	16 16 122 14 2 2 -26 -77 -116 0 -33 -66 82 6 10 -66 -105 0 7 -58

Tabelle 3.2: Beispiele - FV Protokollauszüge

Ein GPS-Datensatz mit je einem Koordinatenpaar wird bei dem FV- und NMEA-Format immer mit einer Frequenz von 1 Hz zur Verfügung gestellt. Die folgenden Typen von Ortungsdaten sind nur im FV-Format enthalten und werden mit einer weitaus höheren Frequenz betrieben, sie liefern somit eine bessere Auflösung und mehr Möglichkeiten bei der Positionsbestimmung.

Odometer / GALA

Außer den GPS-typischen Sensoren besitzt das ODM weitergehende Elemente wie beispielsweise den Radimpulssensor, der auch Odometer genannt wird. Dieser berechnet anhand der Radumdrehungen und des Radumfangs den zurück gelegten Weg und somit auch die aktuelle Geschwindigkeit des Fahrzeugs. Grundsätzlich wurde dieses System als **G**eschwindigkeits**a**bhängige **L**autstärken**a**npassung (GALA) entwickelt um die Lautstärke der vorhandenen Multimediageräte steuern zu können. Außerdem finden diese Daten im Bereich der Telematik eine weitere Anwendung.

Aufgrund der Fähigkeit, die Ortungsdatenmodule von Bosch programmieren zu können, besteht die Möglichkeit, die Frequenz des GALA-Signals auf bis zu 50 Hz zu erhöhen. Dies

[5] Detaillierte Beschreibungen dieser FV-Beispiele befinden sich im Anhang A.2 auf Seite 72.

erhöht die Genauigkeit der Positionsbestimmung der angeschlossenen Datenverarbeitungseinheit (Navigationssystem, Computer).

Gyro

Der Gyro ist ein Drehratensensor, welcher Aufgrund der aktuellen Drehung des Fahrzeugs (bzw. des Ortungsdatenmoduls) den Drehwinkel pro Sekunde (°/s) liefert. Hierbei wird unter Drehung beispielsweise eine Kurvenfahrt angenommen.

Sollte das GPS-Signal einmal ausfallen, sind GALA und Gyro immer noch verfügbar. Somit kann, trotz fehlendem GPS-Signal, die aktuelle Position des Fahrzeugs bestimmt werden. Grundlage hierfür ist der letzte valide GPS-Datensatz sowie die Bewegung ab dieser Position, welche mittels GALA und Gyro bestimmt wird. Wie bei dem Odometer besteht auch bei dem Gyro die Möglichkeit, die Frequenz des Gyro-Daten-Ausstoßes auf bis zu 50 Hz zu steigern.

In den folgenden Abschnitten wird sehr oft der Begriff *GPS-Daten* als Oberbegriff von GPS-, GALA- und Gyro-Daten verwendet.

3.2 Lösungsansätze

3.2.1 Einbetten von GPS-Daten in Einzelbilder

Um Video- und GPS-Daten für eine weitere Verarbeitung zu synchronisieren, gibt es verschiedene Möglichkeiten, bei denen angesetzt werden kann. Eine dieser Möglichkeiten ist die direkte Einbettung der GPS-Daten in das Videobild. Hierbei wird jedes einzelne Frame betrachtet, in seine Grundbestandteile zerlegt und diesen, die gewünschten Informationen hinzugefügt. Das JMF beispielsweise, stellt jedes Videobild in einem *Buffer*-Objekt zur Laufzeit zur Verfügung. Aufgrund der Plug-In-Architektur des Media Frameworks ist es möglich, ein Plug-In (z.B. einen CoDec) zu registrieren, welches jeden *Buffer* in Echtzeit bearbeiten kann. Wie in Abbildung 3.1 dargestellt, besteht ein Videoeinzelbild im prinzipiellen aus einem Byte-Feld, welches die Rot-, Grün- und Blau-Werte eines jeden Bildpunktes in sich trägt.

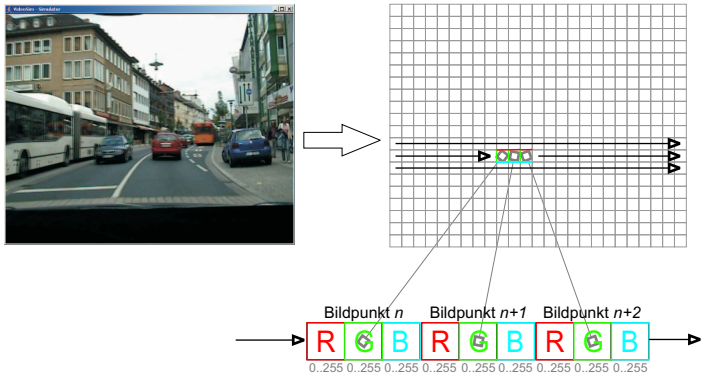


Abbildung 3.1: Kodierung eines Videoeinzelbildes

Die einzelnen Zeilen der in Abbildung 3.1 dargestellten Matrix werden aneinander gehängt. Somit entsteht ein einziges eindimensionales Feld von Bytewerten für ein Videobild. Dieses Byte-Feld kann jedoch um eine gewünschte Anzahl von Elementen erweitert werden. In diesem zusätzlichen Bereich besteht nun die Möglichkeit, die GPS-Daten oder auch andere gewünschte Informationen abzuspeichern. Dies kann wie in Abbildung 3.2 geschehen. Hierbei muss eine entsprechende Codierung der GPS-Daten in einem entsprechenden Format entwickelt werden.

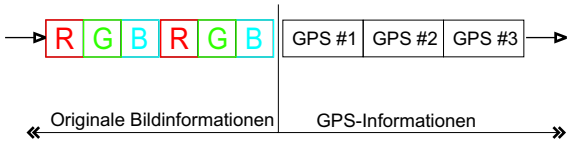


Abbildung 3.2: In ein Bild eingebettete GPS-Daten

Ein Vorteil dieser Methode ist die leichte Synchronisation zwischen jedem Bild des Videos und eines dazu gehörigen GPS-Datensatzes. Dies geschieht im Prinzip automatisch bei der Einbettung, setzt jedoch eine gleiche Frequenz der Video- und GPS-Daten vorraus. Ein Video liegt mit einer fixen Bildrate von 25 Bildern pro Sekunde[6] vor. Das Ortungsdatenmoduls arbeitet aber mit einer Datentransferrate von maximal 50 Hz[7], somit kann eine sehr hohe Auflösung erreicht werden, welche wiederum der doppelten Frequenz des Videos entspricht. Bei einer Einbettung der Ortungsdaten würden folglich Fragmente verloren gehen und nicht mehr beachtet werden. Ein weiterer Nachteil dieses Verfahrens ist die enorme Größe der letztendlich aufgezeichneten Videodatei, da jedem Bild ein fixer Datensatz an Informationen hinzugefügt wird. Ferner besteht die Möglichkeit der Inkompatibilität des erstellten Videos mit gängigen Mediaplayern verbreiteter Betriebssysteme. Ein ebenso

[6] Vgl. 3.1.1 auf Seite 11.
[7] Vgl. 3.1.2 auf Seite 12.

großes Problem besteht in der Komprimierung dieses Videos. Da 12 Minuten 30 Sekunden in etwa 2,6 GB an Festplattenspeicher beanspruchen, ist eine Komprimierung des Videos den unkomprimierten Rohdaten vorzuziehen. Im Rahmen dieser Diplomarbeit wird hierfür die Open-Source Implementierung XviD[8] des MPEG4-Video-Standards eingesetzt. Eine MPEG4 Komprimierung von Videodaten basiert auf dem Erkennen von Objekten innerhalb der Videobilder und der Bewegung dieser Objekte innerhalb des Bildes. Dadurch ist es möglich, lediglich die Veränderungen zwischen Videobildern abzuspeichern und bei der Wiedergabe des Videos diese Informationen wieder zusammen zu fügen. Wären dem Video zusätzliche GPS-Daten jedem Bild hinzugefügt, könnten diese bei der Komprimierung entweder wegfallen (z.B. durch 'Abschneiden' der angefügten Elemente des Byte-Felds) oder den Komprimierungsgrad erheblich verringern. Des Weiteren kann das JMF keine Videodateien größer als 2 GB verarbeiten, was eine Komprimierung zwingend notwendig macht.

3.2.2 Getrenntes Speichern von Video- und GPS-Daten

Das nicht eingebettete Speichern von Daten setzt voraus, diese getrennt auf der Festplatte abzuspeichern. Das Hauptproblem dieser Variante der Fusionierung ist die Synchronisation zwischen Video und GPS-Daten. Die Georeferenzierung eines jeden Videobildes ist Grundvorraussetzung für das 'Wiederfinden' der GPS-Daten in der Fahrsimulation. Es gibt zwei prinzipielle Wege, über die eine Synchronisation vorgenommen werden kann:

- Anhand der Sequenznummer eines jeden Videobildes
- Anhand des Video-Zeitstempels, an dem ein GPS-Signal auftritt

Grundsätzlich spielt es keine Rolle, welche dieser Möglichkeiten zur Anwendung kommt. Im Hinblick auf eine schnelle Realisierung und der leichteren Integrierbarkeit der zweiten Variante wird diese bevorzugt. Bei der hier betrachteten Methode entstehen folglich zwei verschiedene Dateien für verschiedene Aufgabenbereiche. Dennoch besitzen die Inhalte der beiden Dateien Referenzen auf die Inhalte der jeweiligen Pendants. Abbildung 3.3 zeigt den Ablauf der Synchronisation im Detail.

[8] Quelle: [XviD].

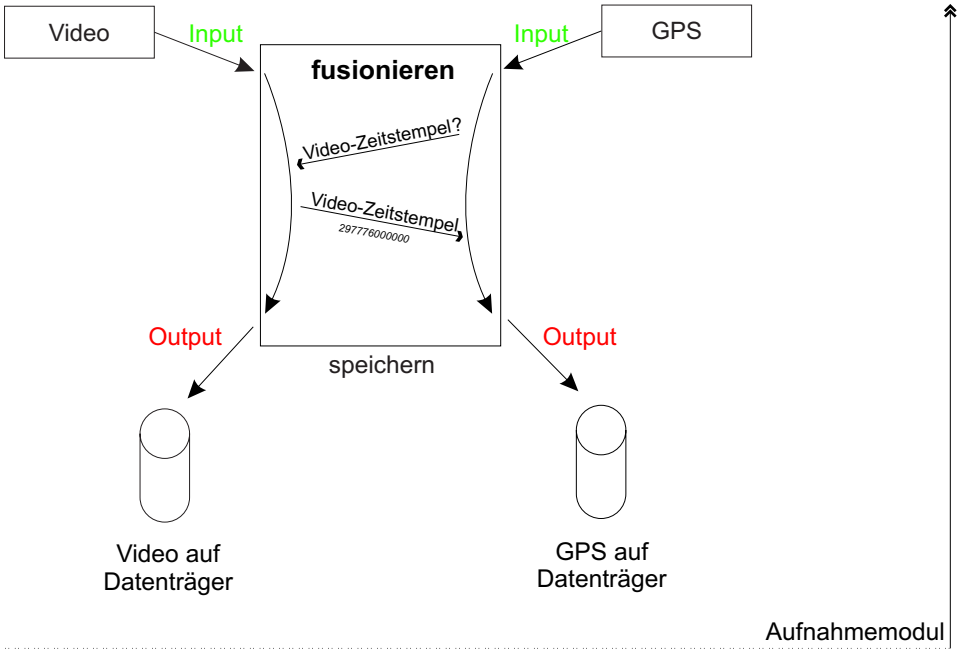


Abbildung 3.3: Synchronisation von Video und GPS

In den nächsten Abschnitten werden verschiedene Ansätze zur getrennten, synchronisierten Speicherung von Video- und GPS-Daten betrachtet und bewertet, um deren Eigenschaften darzustellen, und die Vorteile des letztendlich favorisierten Lösungsansatzes zu zeigen.

Datenbank

Datenbanken (z.B. DB2, Oracle 10g, MS-SQL-Server oder FastObjects) besitzen die Eigenschaft, wichtige Daten konsistent, persistent und höchst performant zugänglich abspeichern zu können. Hierbei ist es egal, ob relationale oder objektorientierte Datenbanken betrachtet werden. Das Grundprinzip ist das gleiche: Abspeichern von Informationen anhand spezifischer Kriterien. Allerdings besteht hier das Problem der Verfügbarkeit. Die relevanten GPS-Daten werden im Fahrzeug während einer Aufnahmefahrt aufgezeichnet. Somit muss die Datenbank auch während dieser Fahrt erreichbar sein, was die Flexibilität des gesamten Systems negativ beeinflusst. Des Weiteren ist der Aufwand/Nutzen-Faktor ein wichtiger Aspekt in der Bewertung. Für die Anbindung einer Datenbank muss eine datenverarbeitende Schicht innerhalb des Programms geschaffen werden, die GPS-Daten über Datenbankbefehle in Echtzeit verarbeitet und diese Daten speichert. Dieser Aufwand ist nicht mit den unter Kapitel 2.1 aufgeführten Anforderungen vereinbar.

XML

Eine andere Möglichkeit der Speicherung von GPS-Daten besteht im Bereich von XML. Dabei wird ein hohes Maß an Interoperabilität erreicht, da es bereits eine Vielzahl von Bearbeitungs- und Lesebibliotheken gibt. Der modulare und standardisierte Aufbau von XML-Dateien ermöglicht die Implementierung schneller Lese- und Schreibroutinen solcher Dokumente. Aufgrund der Standardisierung durch XML-Tags müssen die zu speichernden Daten mit bestimmten Identifikatoren versehen werden. Diese blähen die XML-Dateien bei einem großen Datenumfang stark auf. Eine einminütige Sequenz beispielsweise generiert ca. 80000 einzelne Bytewerte[9], die noch in eine, der Anwendung entsprechenden, Form gebracht werden müssen.

TXT-Datei

Eine weitere Möglichkeit der Synchronisation ohne Einbettung ist das Erstellen einer seperaten Textdatei mit einem kompakten Format zur Datenspeicherung. In ihr werden die GPS-Daten der Reihenfolge nach abgelegt. Zusätzlich werden diese mit einem Zeitstempel des Videostreams versehen, an dem dieser Datensatz auftrat (wie in Abbildung 3.3 dargestellt). Somit ist ein Bezug der GPS-Daten zu der jeweiligen Stelle im Video sichergestellt. Da diese Daten nur mit dem Zeitstempel und nicht mit standardisierten Identifikatoren[10] markiert werden, beschränkt sich der Speicheraufwand auf das Wesentliche: das Ziel der kompakten Datenspeicherung.

3.3 Lösungskonzept der getrennten Speicherung von Video- und GPS-Daten

Die synchronisierte Speicherung von Video- und GPS-Daten setzt voraus, eine der in den Abschnitten 3.2.1 und 3.2.2 vorgestellten Möglichkeiten zu wählen. Die eingebettete Variante des Integrierens der GPS-Daten in die jeweiligen Videobilder erweist sich als die schlechteste aller Optionen. Der benötigte Speicherbedarf sowie der mögliche Verlust von Daten bei einer Komprimierung sind dabei ausschlaggebend. Außerdem ist ein derartig generiertes Video mit Zusatzdaten nicht in allen gängigen Mediaplayern abspielbar. Die nächste Variante wäre das Einlagern der GPS-Daten in eine Datenbank. Da dies kein einbettendes Verfahren ist, ist hier eine Synchronisation von Bild und GPS notwendig. Infolge der nicht erfüllbaren Anforderungen an eine Implementierung mit Hilfe einer Datenbank, kann diese Option vernachlässigt werden. Aufgrund des hohen Speicherbedarfs von XML-Dateien bei den GPS-Datenmengen bleibt nur noch die Möglichkeit diese Daten

[9] Siehe Anhang ?? auf Seite ??.
[10] Wie beispielsweise bei XML.

synchronisiert in einer seperaten Textdatei zu speichern. Ein derartiges Verfahren arbeitet beim Ablegen der Dateien sehr zügig und ist auch im Hinblick auf das spätere Laden dieser Daten sehr praktikabel.

Prinzip: Mit dem aus dem Video stammenden Video-Zeitstempel werden die GPS-Daten im FV-Format markiert:

`[Video-Zeitstempel]:[GPS-Daten im FV-Byte-Format]`

Durch diese Verknüpfung von Video und GPS-Daten, kann einem FV-Datenframe genau eine Stelle in dem korrespondierenden Video zugeordnet werden. Eine solche Zeile mit Zeitstempel und GPS-Daten stellt innerhalb der resultierenden Textdatei eine Zeile dar. Ein neuer Zeitstempel und neue GPS-Daten werden wiederum in einer neuen Zeile zusammengefasst und gespeichert, bis der Aufzeichnungsvorgang der Video- und GPS-Daten beendet wird.

Das FV-Ortungsdatenmodul liefert zwei verschiedene Byte-Frames mit Ortungsdaten. Wie in Tabelle 3.3 beispielhaft dargestellt, werden diese Byte-Frames mit vorangestelltem Video-Zeitstempel[11] in einer Textdatei gespeichert.

Bedeutung	Beispiel
GPS-Positionen inkl. GALA/Gyro	271361000000:51 16 123 15 -1 17 5 7 -44 2 -26 -77 -96 0 79 120 89 0 14 -107 108 0 0 3 52 0 0 0 13 0 0 19 -20 47 1 4 2 -76 0 0 0 -55 0 0 0 -65 -57 -76 6 10 -66 -105 0 6 38
nur GALA/Gyro	271361000000:16 16 122 14 2 2 -26 -77 -116 0 -33 -66 82 6 10 -66 -105 0 7 -58

Tabelle 3.3: Beispiele - Video-Zeitstempel mit zugehörigen FV-Frames

Per Definition handelt es sich dabei um jeweils ein 51-Byte-Frame (oben) und ein 16-Byte-Frame (unten). Zu Beginn einer jeden Zeile steht eine lange Zahl, welche den Zeitstempel[12] des Auftretens des Datensatzes im Video repräsentiert. Dieser ist relativ zum Start der Videoaufnahme bei 0 Sekunden. Nach dem Doppelpunkt folgen die GPS-Daten im Rohformat, jedes einzelne Byte ist durch ein Leerzeichen getrennt. Sehr auffällig ist hier die Tatsache, dass auch negative Werte auftreten. Wie in der Spezifikation des FV-Datenformats angegeben, werden diese Daten als einzelne Bytes übertragen, deren Wertebereich in den Grenzen von 0 bis 255 liegt. Aufgrund der Verwendung von Java und dessen Bytekodierung (-128 bis +127) werden teilweise auch negative Werte abgespeichert. Nichtsdestotrotz können diese Werte 'normal' weiterverarbeitet werden, da es sich hier nur um eine andere Präsentationsform handelt und die Daten in ihrem Wesen nicht verändert werden. Ein 'kleines', so genanntes 16-Byte-Frame, liefert in diesem Fall die Daten des GALA und des Gyro, also die Daten von dem Drehraten- sowie dem Radumlaufsensor.

[11] Trennung zwischen Video-Zeitstempel und FV-Datenframe durch einen „:“.
[12] Hier ein Long-Wert, gezählt in Millisekunden.

Wie in Abschnitt 3.1.2 beschrieben, werden diese Daten mit einer Frequenz von 50 Hz geliefert. Im Gegensatz hierzu kann das 51-Byte-Frame nur einmal pro Sekunde, also mit 1 Hz, empfangen werden. In diesem sind außer den GALA- und Gyro-Daten, die *normalen* GPS-Daten enthalten, wie z.B. UTC-Zeit, Ortskoordinaten, Geschwindigkeit, Richtung, Höhe über Normal Null und andere, wie in Abschnitt A.2 angegeben. Zusätzlich fällt auf, dass der Zeitstempel bei beiden Frames hier gleich ist, was sich in der hohen Ausstoßfrequenz der GALA- und Gyro-Daten (50 Hz) begründet. Anscheinend wurde hier gerade ein 16- und 51-Byte-Frame annähernd zeitgleich gesendet[13].

Parallel zum Aufzeichnen dieser Daten wird der Videostream in einer AVI-Datei[14] abgelegt. Somit ist ein hohes Maß an Flexibilität gewährleistet. Hilfreich ist in diesem Bereich auch eine eindeutige Kennzeichnung der beiden entstandenden Dateien im Dateinamen. Dies kann den Bezug zwischen Video und GPS für den Nutzer und das Simulationssystem kenntlich machen. Die Synchronisation der Video- und GPS-Daten innerhalb dieser beiden Dateien geschieht durch den Zeitstempel des Videos. Tritt zu einer bestimmten Zeit 'x' (relativ zum Beginn des Videos) ein GPS-Datensatz auf, dann wird der Zeitstempel des Zeitpunkts 'x' aus dem Video angefordert und dieser GPS-Datensatz mit diesem Zeitstempel markiert. Somit ist eine eindeutige Zuordnung des Datensatzes zu einer Position in dem dazu gehörigen Video gewährleistet.

[13] Im Vergleich zu dem Ortungsdatenmodul hat das Video lediglich eine Frequenz von 25 Hz und die Zeitstempel werden immer nur von einem Bild (*Buffer*-Objekt) aufgenommen.

[14] Gemeinsame Kompression von Sprache und Bildern von Microsoft.

4 Präsentation von raum- und videobezogenen Daten

4.1 Problematik

Die Präsentation von raum- und videobezogenen Daten setzt auf zuvor aufgezeichneten Video- und GPS-Daten auf[1]. Diese Daten bilden die Grundlage für ein Simulationsmodul in dem sich der Fahrer der Fahrsimulation bewegen kann. Durch interaktive Einflussnahme des Fahrers auf die Präsentation (wie beispielsweise das Beschleunigen) soll ihm ein gewisses Maß Freiheit gegeben werden. Dies kann jedoch nur unter gewissen Randbedingungen erfolgen, da stets eine Beschränkung auf das aufgenommene Video besteht.

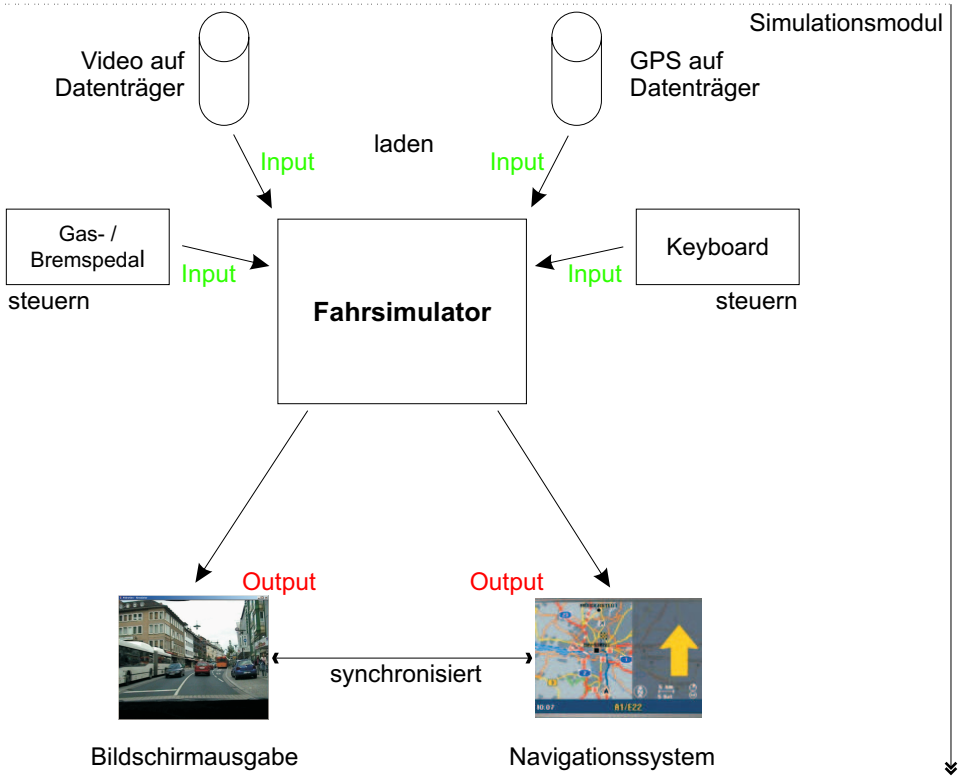


Abbildung 4.1: Anforderungsvisualisierung

[1] Vgl. Abschnitt 3.3 auf Seite 18.

Der in Abbildung 4.1 gezeigte Systemüberblick über das Simulationsmodul wird in den folgenden beiden Abschnitten genauer analysiert.

4.1.1 Eingangsparameter

Alle in Abbildung 4.1 mit *Input*[2] bezeichneten Größen sind als Eingangsparameter definiert. Hierbei kann in zwei Klassen unterschieden werden:

- statische Eingangsparameter (Video- und GPS-Daten auf dem Datenträger)
- dynamische Eingangsparameter (Gas-/Bremspedal- und Keyboardsteuerung)

Die statischen Eingangswerte repräsentieren die Videodaten als AVI-Datei sowie die GPS-Daten als Textdatei[3].

4.1.2 Ausgangsparameter

Alle in Abbildung 4.1 mit *Output*[4] bezeichneten Größen sind als Ausgangsparameter definiert. Diese ergeben sich aus der Verarbeitung der Eingangsparameter. Prinzipiell werden die beiden Eingangswerte, Video und GPS, durch den Fahrsimulator synchronisiert ausgegeben, worauf weiterhin noch die beiden anderen, als Gas-/Bremspedal und Keyboard gekennzeichneten, Eingangssignale wirken. Sie verändern und manipulieren die Dartstellung der statischen, synchronisierten Eingangsparameter in Richtung einer dynamischen Fahrsimulation.

[2] In Abbildung 4.1 grün hervorgehoben.
[3] Vgl. Abschnitt 3.2.2 auf Seite 16*ff*.
[4] In Abbildung 4.1 rot hervorgehoben.

4.2 Entwicklung eines Lösungskonzepts

4.2.1 Videokomponente

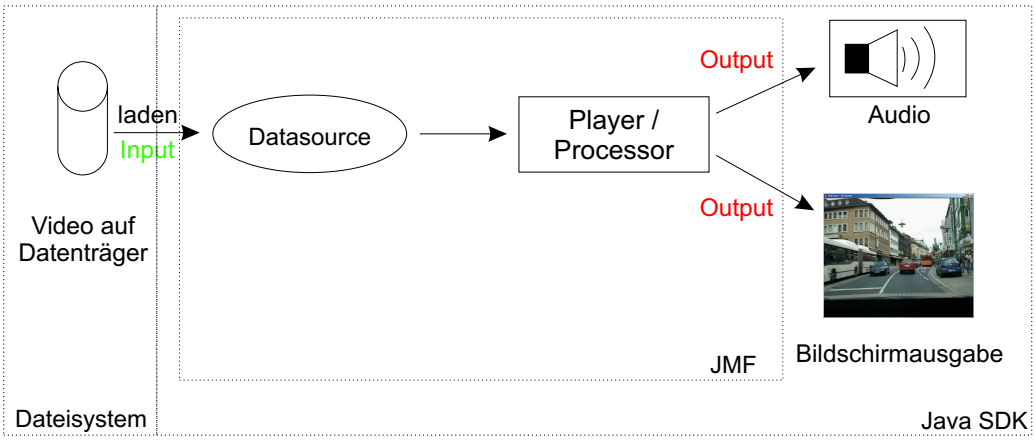


Abbildung 4.2: Verarbeitung der Videodaten

Abbildung 4.2 zeigt den Verarbeitungsprozess der Videodaten. Diese befinden sich, wie in Abschnitt 3.2.2 vorgestellt, auf einem Datenträger (z.B. lokale Festplatte) und werden über das Java SDK und das JMF[5] der Fahrsimulation zur Verfügung gestellt. Grundsätzlich gibt es zwei Stufen, die ein Video innerhalb des Media Frameworks durchlaufen muss, bevor es den Status der 'Abspielbarkeit' erreicht. Wie in Abbildung 4.2 zu sehen, wird zunächst aus dem Video eine *Datasource*[6] erzeugt, welche entweder durch eine Datei (MediaLocator) oder eine URL definiert ist.

Sie verwaltet eine Menge[7] von SourceStream Objekten. Diese SourceStream Objekte repräsentieren die Quellen jeglicher, im JMF verarbeitbarer Daten. Aus einer *Datasource* wird wieder ein *Player* oder *Processor* generiert. Der *Player* ist ein Objekt zum Visualisieren von multimedialen Daten. Dabei besitzt ein *Processor* die gleichen Funktionen wie ein *Player*, ist jedoch um Komponenten erweitert. Unter anderem kann er selbst eine neue *Datasource* generieren, welche wiederum durch ein anderes Multimediaprogramm (beispielsweise auch auf Basis des JMF) genutzt werden kann. Des Weiteren bietet ein *Processor* die Möglichkeit, wie in Abschnitt 2.1.2 dargestellt, Plug-Ins (z.B. Codecs oder VideoRenderer) einzubinden. Die Hauptfunktionalität im Kontext der Fahrsimulation ist die Möglichkeit der Präsentation von Bewegtbildern in einer GUI. Die multimedialen Inhalte bilden die Schnittstelle zum Fahrer und spiegeln die Human-Machine-Interaction[8] wieder.

[5] Vgl. Abschnitt 2.1.4 auf Seite 7.
[6] Vgl. Abschnitt *DataModel* in [JMF-Guide].
[7] Im Sinne der mathematischen Mengenlehre.
[8] Vgl. Abschnitt 1.1 auf Seite 1.

In diesem Bereich definiert das JMF zwei essentielle Komponenten, die zur Präsentation und Steuerung des Videostroms dienen.

1. VisualComponent
2. ControlComponent

Die *VisualComponent* stellt der GUI die Bilddaten für die Darstellung zur Verfügung. Das JMF kann hierbei angewiesen werden, eine Heavyweight- oder Lightweight-Component[9] zu liefern. Dadurch kann festgelegt werden, ob das Videobild in einem AWT-Element oder in einem Swing-Element innerhalb der Java-Benutzeroberfläche angezeigt wird. Daraus ergeben unterschiedliche Eigenschaften im Bezug auf die Präsentation des Videos in Verbindung mit der Performance und Priorisierung der Darstellung. Die zweite Komponente in diesem Bereich ist die *ControlComponent*. Sie ermöglicht viele Funktionen, die zur Steuerung eines Videos nötig sind (beispielsweise Starten, Stoppen und Vorspulen des Videos).

4.2.2 GPS-Komponente

Parallel zur Darstellung des Videos werden auch die GPS-Daten dynamisch im Speicher gehalten. Zu jedem Zeitpunkt innerhalb des Videos liegt ein GPS-Datensatz vor, der abgerufen wird. Gerade in Bezug auf die Anbindung von Navigationssystemen an die Fahrsimulation ist dies ein wichtiger Aspekt.

Es befinden sich diese Daten zu Beginn einer jeden Fahrsimulation in einer Textdatei[10] (mit einer zugehörigen Videodatei). Diese Datei wird von Anfang bis Ende eingelesen und die darin enthaltenen Daten werden in eine dynamische Datenstruktur zur flexiblen, komfortablen und höchst performanten Verarbeitung gebracht.

[9] Vgl. 'Heavyweight vs. Lightweight'-Abschnitt im Anhang B.2.2 auf Seite 79.
[10] Vgl. Abschnitt 3.3 auf Seite 18.

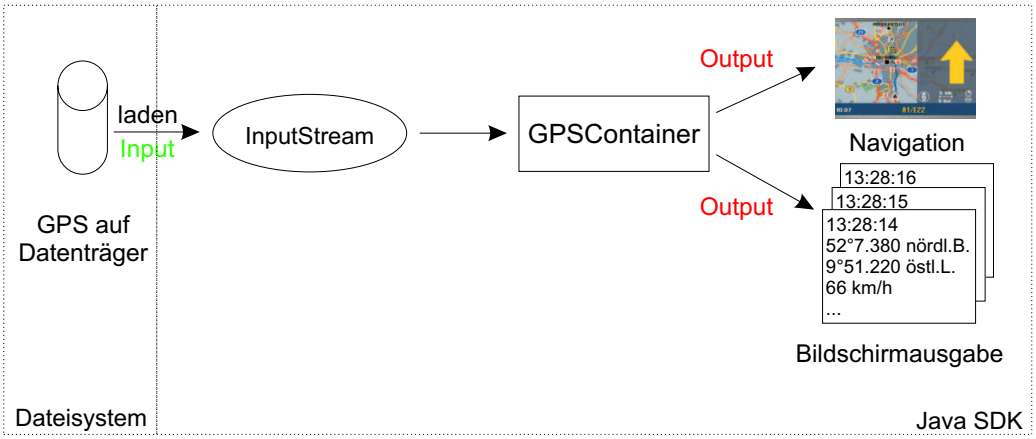


Abbildung 4.3: Verarbeitung der GPS-Daten

Abbildung 4.3 zeigt die prinzipielle Verarbeitungskette der GPS-Daten, beginnend bei der Datei auf einem Datenträger bis hin zur Darstellung im Rahmen der Fahrsimulation und/oder in einem Navigationssystem. Die zentrale dynamische Softwarekomponente in diesem Bereich ist der *GPSContainer*, welcher die Daten zur Laufzeit performant zur Verfügung stellt.

Analog zu dem FV-Ortungsdatenmodul liefert die Fahrsimulation Daten mit einer Frequenz von 50 Hz, da zu jeder Sekunde in dem Videostrom ca. 50 Datensätze der GPS-, Gyro- und GALA-Daten aufgezeichnet wurden[11]. Ein komplettes GPS-Datenframe umfaßt 51 Byte und steht nur einmal pro Sekunde zur Verfügung. Die restlichen 49 Datenframes bestehen aus nur 16 Byte mit Gyro- und GALA-Werten (diesem fehlen die Ortskoordinaten aus der Bestimmung durch das Global Positioning System). Infolgedessen muss eine Differenzierung in der dynamischen Datenhaltung derartiger Werte möglich sein. Eine grafische Darstellung dieser Datenrate ist in Abbildung 4.4 zu sehen.

[11] Vgl. Aschnitt 3.1.2 auf Seite 12.

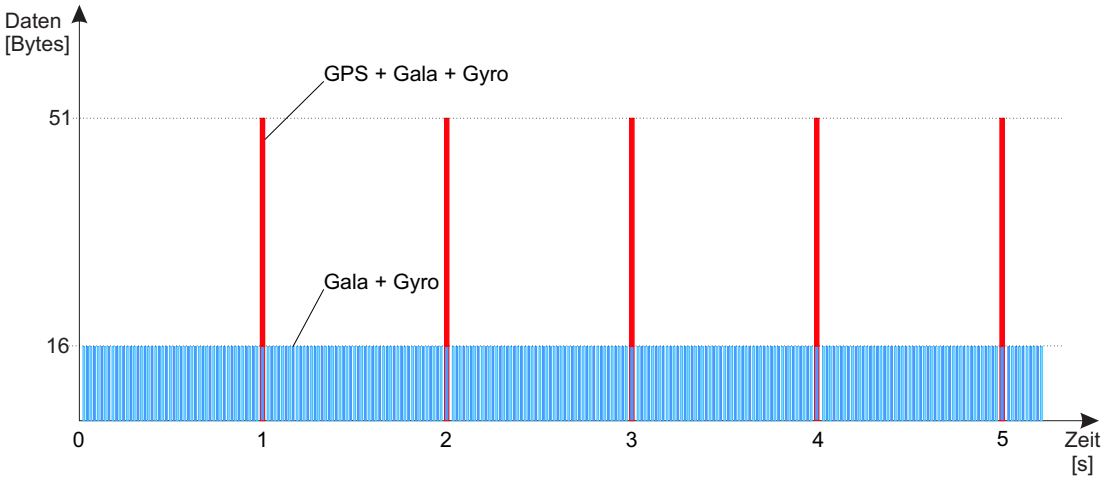


Abbildung 4.4: GPS-, Gyro- und GALA-Datenintervall

Die 51-Byte-Frames enthalten alle Gyro- und GALA-Werte der 16-Byte-Frames sowie zusätzlich die GPS-Koordinatendaten. Demzufolge ist ein 16-Byte-Frame, im mathematischen Sinne, eine Untermenge eines 51-Byte-Frames. Aufgrund dieser speziellen Eigenschaften der Daten beinhaltet also ein 51-Byte-Frame auch die Daten eines 16-Byte-Frames. Somit spiegelt sich die konzeptionelle Verbundenheit auch in der Datenmodellierung wider.

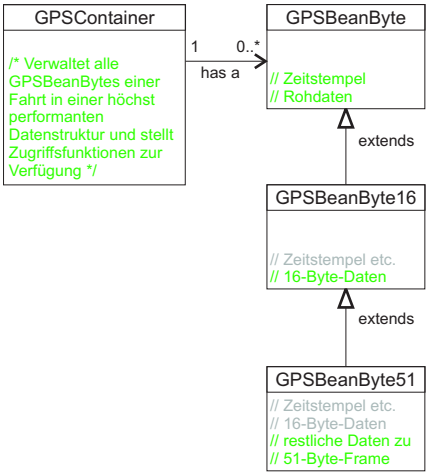


Abbildung 4.5: GPS-Datenmodellierung

Abbildung 4.5 zeigt die Modellierung der GPS-Daten in UML. GPSBeanByte16 und GPSBeanByte51 kapseln in diesem Fall die jeweiligen Daten eines 16- sowie eines 51-Byte-Frames. In diesen, zur Laufzeit erstellten Objekten, werden die Daten im Rohfor-

mat (original FV-Byteformat[12]) sowie in aufbereiter, schnell verfügbarer Form gehalten, um beispielsweise durch einen einzigen Aufruf die aktuelle Geschwindigkeit des Fahrzeugs abzufragen. Des Weiteren werden auch zur Zeit der Erstellung eines solchen GPS-Datenobjekts die Daten in das NMEA-Format[13] konvertiert und dynamisch vorgehalten. Somit können die GPS-Daten bei Bedarf zu jeder Zeit abgefragt und beispielsweise an Navigationsgeräte[14] geliefert werden.

4.2.3 Zusammenwirken von Video und GPS

Das Zusammenwirken von Video und GPS spiegelt die synchronisierte Ausgabe des Videostroms mit den zugehörigen Positionsdaten an jedem Punkt der Fahrt wieder. Bei einer Videofahrt treten die Geoinformationsdaten stets sequentiell auf. Wird die Fahrsimulation mit dem entsprechenden Video gestartet, werden die Daten über die Position des Fahrzeugs nacheinander benötigt. Hierzu wird mit dem Start des Videos eine Thread-Instanz gestartet, die in einer definierten Frequenz (z.B. 50 Hz) den aktuellen Zeitstempel des Videos abfragt. Aufgrund der synchronisierten Speicherung[15] von Video- und GPS-Daten[16] können diese beiden Datensätze (Video und GPS) so parallel abgearbeitet werden. Mit jedem Fortschreiten des Videos ändert sich der Zeitstempel der aktuellen Position innerhalb der Fahrsimulation.

[12] Vgl. Abschnitt A.2 auf Seite 72.
[13] Vgl. Abschnitt A.1 auf Seite 70.
[14] Auch an alle GPS verarbeitendenden Geräte, die Daten im NMEA-Format interpretieren können.
[15] Vgl. Abschnitt 3.3 auf Seite 18.
[16] Wurden mit ca. 50 Hz aufgezeichnet, vgl. Abschnitt 3.1.2 auf Seite 12.

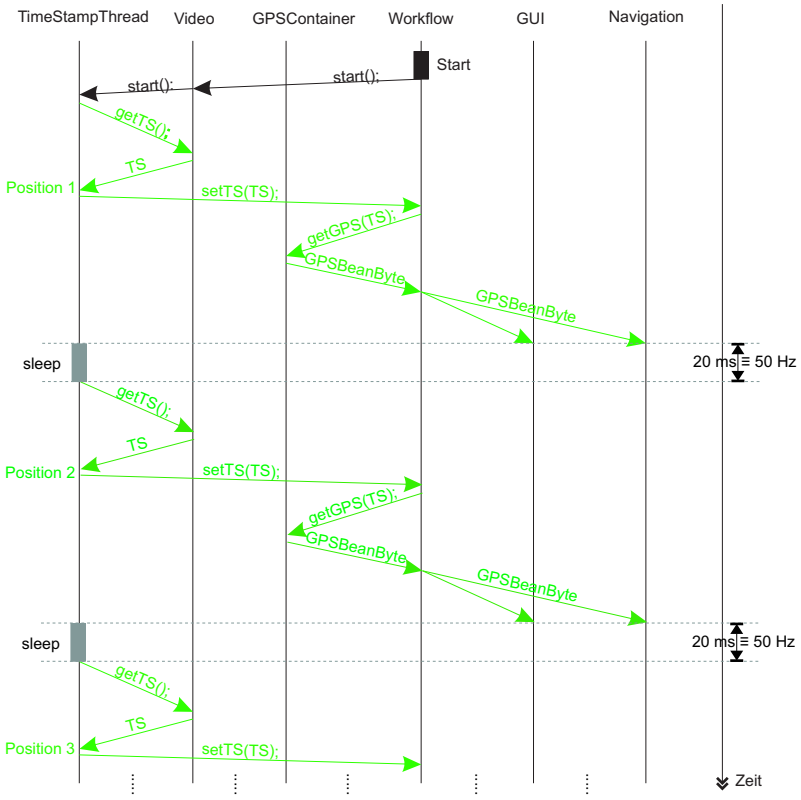


Abbildung 4.6: GPS/Video Synchronisation

Abbildung 4.6 zeigt die interne Synchronisation von Video- und GPS-Daten. Ist der Zeitstempel der aktuellen Position des Videos ermittelt, wird er an das *Workflow* (Verarbeitende Komponente) weitergereicht. Von dort aus werden die GPS-Daten von dem *GPSContainer* (vgl. Abbildung 4.5) der geladenen Fahrt abgefragt. Aufgrund der in Abschnitt 3.3 auf Seite 18 dargestellten Methode zum synchronisierten Speichern sind die GPS-Daten zu einem gegebenen Zeitstempel (Position in dem Video) schnell verfügbar. Ist nicht der genaue Zeitstempel der aktuellen Position des Videos in dem Datensatz vorhanden, wird 'der nächst bessere' betrachtet. Der dadurch auftretende Effekt der Rundung kann in diesem Fall vernachlässigt werden, da bei einer Frequenz von 50 Hz so viele Daten vorliegen, dass die Ungenauigkeit in der Positionsbestimmung gegen 0 approximiert. Diese GPS-Daten, durch eine *GPSBeanByte* (vgl. Abbildung 4.5) repräsentiert, werden dann an die GUI- und Navigationskomponente weitergereicht, die diese Daten weiterverarbeiten kann.

4.3 Steuerung der Fahrsimulation

4.3.1 Konzeptionelle Steuerungsmöglichkeiten

Ein Video kann als ein Fluss von Daten betrachtet werden[17]. Demzufolge ist die Steuerung einer Fahrt durch dieses Video eingeschränkt. Beispielsweise sind der unerwartete Wechsel der Fahrspur oder das Abbiegen auf eine Straße, die nicht vorgesehen und aufgezeichnet ist, in dieser Ausprägung einer Fahrsimulation nicht möglich. Einzig die Manipulation der Abspielgeschwindigkeit des Videos ist beeinflußbar. Die Videoabspielrate zur Manipulation der Abspielgeschwindigkeit liegt in einem Wertebereich beginnend bei 0% über 100% bis zu einer durch die Hardware definierten Obergrenze. 100% entprechen also in diesem Kontext genau der Abspielgeschwindigkeit des Videos, mit dem es aufgenommen wurde. Die, durch die Hardware definierte Obergrenze wird durch die Geschwindigkeit der CPU und des gesamten Systems festgelegt. Da ein Video mit 25 Bildern pro Sekunde aufgenommen ist, ändert sich diese Bildrate entsprechend der Manipulation der Videoabspielrate. Bei der doppelten Abspielgeschwindigkeit des Videos (200% Videorate), würde sich demzufolge auch die Bildrate auf 50 Bilder pro Sekunde verdoppeln. Aufgrund der Komprimierung des Videos mit dem MPEG4-Kompressionsverfahren müssen in diesem Fall alle Bilder betrachtet werden. Folgende Abspiel- und Bildraten sowie CPU-Auslastungen[18] ergeben sich hierdurch.

Videorate	Bildrate (theor.)	CPU-Last
0% = 0.0	0 Bilder/Sekunde	~ 0%
50% = 0.5	12.5 Bilder/Sekunde	~ 15%
100% = 1.0	25 Bilder/Sekunde	~ 42%
200% = 2.0	50 Bilder/Sekunde	~ 95%
220% = 2.2	55 Bilder/Sekunde	~ 100%
240% = 2.4	60 Bilder/Sekunde	~ 100%
240+% = 2.4+	60+ Bilder/Sekunde	~ 100%

Tabelle 4.1: Videoabspiel- und Bildraten im Verhältnis

Ab einem gewissen Wert, wie in Tabelle 4.1 dargestellt, steigt bei der Erhöhung der Videoabspielrate die Auslastung der CPU auf 100%, dem Maximalwert. Dies bedeutet, es werden zu viele Bilder pro Sekunde betrachtet. Im Falle eines mit MPEG4 komprimierten Videos müssen diese Bilder somit auch schneller dekomprimiert werden. Die Hardware ist nicht mehr in der Lage, alle Bilder darzustellen und verwirft diese. Daraus entsteht eine sehr starke Beeinträchtigung in der Visualisierung des Videos. Somit wird eine Obergrenze für die manipulierte Abspielrate bei 1.9 (190%) festgelegt.

[17] Vgl. Abschnitt 3.1.1 auf Seite 11.
[18] Basierend auf Feldversuchen mit einer im Anhang C.4 auf Seite 84 beschriebenen Konfiguration.

Eine dynamische Anpassung der Abspielrate des Videos an die wahre Geschwindigkeit des sich im Video bewegenden Autos ist, aufgrund der technischen Randbedingungen, nicht möglich. Es existieren zwei Eingangsgrößen die dabei betrachtet werden:

- Abspielrate des Videos
- Geschwindigkeit des Fahrzeugs (GPS oder GALA)

Nun wird eine Funktion definiert, die die Geschwindigkeit des Fahrzeugs auf die des Videos abbildet. Dies bedeutet, dass Aufgrund einer Geschwindigkeitseinstellung des Fahrers der Fahrsimulation (z.B. 60 km/h) die Abspielrate des Videos mit einer hohen Frequenz verändert werden müsste. Da die angegeben 60 km/h gehalten werden müssen, sich aber die Geschwindigkeit des Fahrzeugs, also des Fortbewegens im Video ständig ändert, muss somit zu jeder Geschwindigkeitsänderung in der Fahrt (GPS- oder GALA-Geschwindigkeit) die Abspielrate des Videos angepasst werden. Für dieses Anpassen der Abspielrate wird jedoch intern das Video kurzzeitig angehalten, die neue Abspielrate wird gesetzt und das Video wird wieder gestartet. Dieser Vorgang mit einer zu hohen Wiederholrate kann jedoch von der Videokomponente nicht verarbeitet werden. Des Weiteren besteht die Möglichkeit des Überschreitens der definierten Obergrenze für die Videoabspielrate. Würde sich das Fahrzeug im Video mit 10 km/h fortbewegen und der Fahrer der Fahrsimulation die Geschwindigkeit, mit der er sich fortbewegen möchte, auf 60 km/h festsetzen, würde dies dem Faktor 6.0, also 600% Prozent, entsprechen. Der Fahrsimulator müsste nach einer aus Tabelle 4.1 abgeleiteten Linearisierung[19] 150 Bilder pro Sekunde berechnen und darstellen.

4.3.2 Input für Geschwindigkeitssteuerungen

Als Steuerungsgeräte für die Beeinflussung der Abspielgeschwindigkeit des Videostroms stehen die Tastatur des Computers sowie diverse Game-Controller zur Verfügung. Über die Tastatur kann mit Hilfe der Cursor-Tasten die Geschwindigkeit manipuliert werden. Ein Game-Controller wie beispielsweise das „Microsoft SideWinder ForceFeedback Wheel“ besitzt im Gegenzug ein Gas- und Bremspedal sowie ein Lenkrad mit mehreren Funktionstasten, die frei programmierbar sind. Von den Gas-/Bremspedalen sowie dem Lenkrad können die aktuellen Werte ausgelesen werden. Dabei liegen alle drei Elemente in einem Koordinatensystem mit drei Achsen, wobei folgendes gilt:

[19] Vgl. Abschnitt direkte Proportionalitäten im Anhang C.3 auf Seite 83.

Funktion	Achse
Lenkrad	X-Achse
Gaspedal	Y-Achse
Bremspedal	Z-Achse

Tabelle 4.2: Achsenaufteilung „Microsoft SideWinder ForceFeedback Wheel“

Die Wertebereiche dieser drei Achsen sind jeweils mit $[-1.0, +1.0]$ festgelegt. Doch die Ausgangspositionen der X-, Y- und Z-Achsen unterscheiden sich. Die X-Achse, also das Lenkrad, ist in der Ausgangsstellung bei 0.0. Dies bedeutet, dass das Lenkrad auf eine Geradeausfahrt eingestellt ist. Somit besteht noch weiterhin die Möglichkeit nach links oder rechts einen maximalen Einschlag von -1.0 (links) oder +1.0 (rechts) zu verarbeiten. Im Gegensatz dazu befindet sich die Ausgangsstellung der Y- und Z-Achse, also des Gas- und Bremspedals, bei +1.0. Die maximale Ausdehnung diese Pedale beträgt -1.0. Daraus ergibt eine maximale Änderung der Pedalstellung von 2.0 Einheiten, was bei dem Gaspedal mit Vollgas und bei dem Bremspedal mit einer Vollbremsung zu interpretieren ist. Abbildung 4.7 zeigt die dargestellten Achsen, deren Wertebereiche sowie die Ausgangswerte der jeweiligen Achsen.

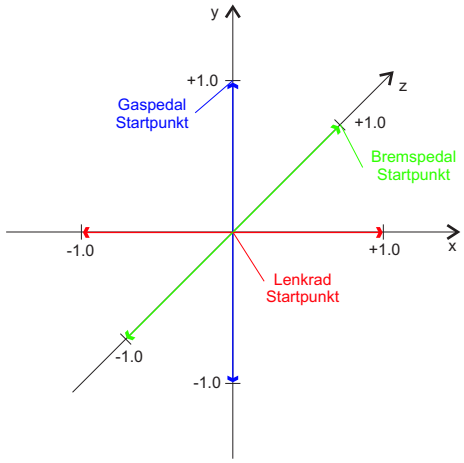


Abbildung 4.7: Achsenaufteilung „Microsoft SideWinder ForceFeedback Wheel“

4.4 Anbindung an bestehende Navigationssysteme

4.4.1 Problematik und Zielsetzung

Die in Abbildung 4.3 auf Seite 25 dargestellte Verarbeitung der GPS-Daten besteht zum Teil auch aus der Anbindung an bestehende Navigationssysteme. Der Kontext der Aufgabenstellung sieht Versuchsreihen mit Probanden vor. Anhand derer soll gezeigt werden, in

wie weit die detaillierte Darstellung von Navigationssystemen den Fahrer in seiner eigentlichen Aufgabe, dem Führen des Fahrzeugs, ablenkt[20]. Dabei wird davon ausgegangen, dass eine bestimmte Strecke aufgenommen wurde, wie sie ein Navigationssystem auch berechnen würde. Die Fahrsimulation ersetzt reale Fahrten. Während der Fahrt soll es möglich sein, die Navigationsausgabe im Cockpit anzuzeigen. Die im Rahmen dieser Diplomarbeit entwickelte Fahrsimulation muss demzufolge auch ein Modul enthalten, welches in der Lage ist, GPS-Informationen an Navigationssysteme zu liefern.

Anbindbare Systeme

Neben konventionellen Hardware-Navigationssystemen von Blaupunkt, wie beispielsweise dem TravelPilot, existieren weitere Softwarelösungen, die hier auch betrachtet werden. Auf der einen Seite gibt es die „OffBoardNavigation“, welches ein Navigationssystem in Software ist. Des Weiteren steht ein Programm „SensorTool“ zum Testen der verschiedensten Formate und Schnittstellen zu einem Ortungsdatenmodul zur Verfügung. Ein Navigationssystem verhält sich passiv und nimmt GPS-Daten von externen Geräten (wie dem Ortungsdatenmodul) auf. Die Fahrsimulation emuliert ein derartiges Ortungsdatenmodul, indem es die gleichen Protokolle, Formate und Schnittstellen verwendet, um GPS-Informationen dem Navigationssystem zur Verfügung zu stellen.

Schnittstellen

Die Anbindung an derartige Systeme ist auf Basis von zwei Schnittstellen möglich:

1. Netzwerkschnittstelle
2. Serielle Schnittstelle

Erstere Schnittstelle offeriert die Möglichkeit der Anbindung eines Ortungsdatenmoduls über ein Netzwerk. Hierbei wird das TCP/IP - Protokollpaket verwendet und ermöglicht die Anbindung über ein Local Area Network. Die zweite Schnittstelle im Bereich der Kopplung von Ortungsdatenmodul und Navigationssystemen ist die serielle Schnittstelle (in Bezug auf die „OffBoardNavigation“ ist dies z.B. einer der COM-Ports).

Formate

Es existieren zwei grundsätzliche Formate zur Übertragung von Ortungsdaten eines Ortungsdatenmoduls an Verarbeitungseinheiten (Navigationssysteme, Computer). Der offene NMEA-0183[21] Standard, sowie das interne FV-Format[22] der Firma Bosch.

[20] Vgl. Abschnitt 1.2 auf Seite 1.
[21] Vgl. Abschnitt A.1 auf Seite 70.
[22] Vgl. Abschnitt A.2 auf Seite 72.

4.4.2 Anbindung

Zusammenfassend existieren also zwei Schnittstellen (Netzwerk- und serielle Schnittstelle), an die GPS-Daten geschrieben werden. Jede dieser Schnittstellen unterstützt jeweils zwei Formate (NMEA und FV), daraus resultieren vier Sendeverfahren. Im Hinblick auf eine Erweiterung der Schnittstellen oder der Formate ist eine flexible Gestaltung des Moduls zur Anbindung an externe Systeme vorteilhaft. Grundsätzlich gibt es zwei Varianten zum Senden von GPS-Daten an ein Ziel:

1. im NMEA-Format
2. im FV-Format

Um diese beiden Funktionalitäten zu erreichen, wird folgende Systemstruktur vorgeschlagen:

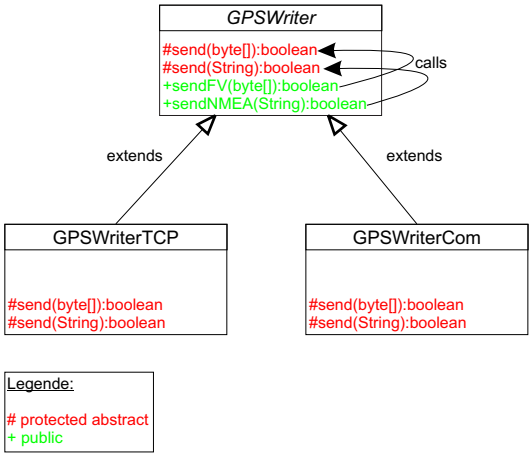


Abbildung 4.8: GPSWriter-Architektur

Abbildung 4.8 zeigt die interne Struktur des GPS-Sendevorgangs. Hierbei dient der *GPSWriter* als abstrakter Wrapper zum Kapseln der Funktionalität, die die beiden public-Methoden[23] *sendFV()* und *sendNMEA()* bereitstellen sollen. Diese rufen intern die jeweiligen protected-Methoden *send()* zur Kapselung auf. Dadurch wird eine Funktionalität in der Superklasse (*GPSWriter*) zur Verfügung gestellt, die erst in der Subklasse (hier *GPSWriterTCP* und *GPSWriterCom*) implementiert wird. Dieses Verfahren hat den Vorteil einer flexiblen Struktur und bietet somit die Möglichkeit, die Funktionalitäten schnell auf sich ändernde Rahmenbedingungen anzupassen (wie zum Beispiel die Integration von Bluetooth als eine weitere Schnittstelle). Abbildung 4.9 zeigt die schnittstellenorientierte Anbindung von externen, GPS-fähigen Geräten, wie beispielsweise Navigationssysteme.

[23] Grün hervorgehoben.

GPS- und Videodaten werden von dem Datenträger geladen, abgespielt und synchronisiert[24]. Zu jeder vorhandenen Position werden GPS-Daten (bei der Aktivierung dieser Funktion) an die entsprechenden Schnittstellen gegeben und ausgesandt. Diese Daten können dann von einem externen Gerät aufgenommen und verarbeitet werden. Grau hinterlegt ist eine mögliche Erweiterung um die Bluetooth-Schnittstelle, um die Flexibilität in der Softwarekonzeption zu verdeutlichen.

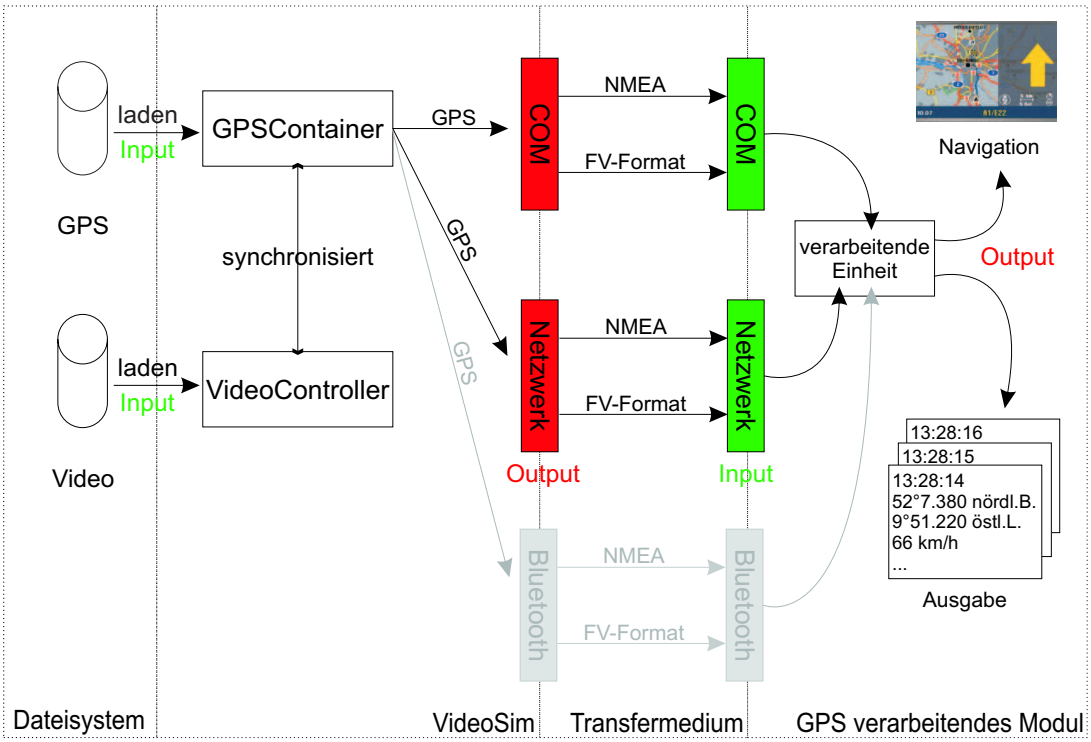


Abbildung 4.9: Anbindungsstruktur von externen Geräten

[24] Vgl. Abschnitt 4.2.3 auf Seite 27.

5 Fusion von mehrdimensionalen raum- und videobezogenen Daten

5.1 Projekte, Strecken und Fahrten

Mehrdimensionale raum- und videobezogene Daten ergeben sich aus dem Kontext der Diplomaufgabenstellung. Obwohl eine Beschränkung der Fahrsimulation auf das aufgenommene Video besteht, eröffnet dies die Möglichkeit eine Strecke der Fahrsimulation mehrmals aufzunehmen. Die daraus entstehenden Daten müssen in einer flexiblen Datenstruktur zur Verarbeitung innerhalb der Fahrsimulation verwaltet werden. Eine Gliederung der sich daraus ergebenden GPS- und Videodaten sowie deren Zusammenfassung zu verschiedenen Gruppen kann wie folgt definiert werden:

Strecke (Track):

Eine Strecke repräsentiert einen Weg von dem Startpunkt A zum Zielpunkt B über eine festgelegte Route. Diese Route ist meist so gewählt, wie sie ein Navigationssystem bei einer Navigationsanfrage berechnen würde. Die Strecke kann mehrere unterschiedliche Fahrten enthalten und ist immer Teil eines Projekts.

Fahrt (Run):

Eine Fahrt ist immer ein Teil einer Strecke und spiegelt die Aufnahmefahrt einer festgelegten Strecke wieder. Fahrten unterscheiden sich beispielsweise in den Fahr- und Wettersituationen. Mehrere Fahrten einer Strecke können eine Tagfahrt, eine Nachtfahrt und eine Regenfahrt sein. Sie enthalten jeweils die Video- und GPS-Daten für eine aufgenommene Fahrt einer Strecke.

Projekt (Project):

Ein Projekt kann eine Vielzahl von Strecken umfassen und in der Fahrsimulation geladen werden. Es gibt an, welche Strecken gefahren werden und wie die Reihenfolge dieser Strecken ist. Eine XML-basierte VSX-Datei[1] speichert alle benötigten Daten zu einem Projekt inklusive der Strecken- und Fahrteninformationen.

[1] Vgl. mit VSX Spezifikation in Anhang A.3 auf Seite 72.

Diese hierarchische Struktur innerhalb der Fahrsimulation wird in Abbildung 5.1 in einem UML-Klassendiagramm[2] modelliert. Sie zeigt das *Workflow*, welches die verwaltende Komponente in der Fahrsimulation ist. Besitzt das *Workflow* eine Assoziation zu einem Projekt (*Project*), wurde ein Projekt in der Fahrsimulation geladen. Ein Projekt kann auf mehrere Strecken (*Track*) verweisen, die wiederum Assoziationen zu mehreren Fahrten (*Run*) dieser Strecke haben können. In umgekehrter Reihenfolge besitzt jede Fahrt eine existenzabhängige Aggregation zu der dazu gehörigen Strecke, diese zu dem dazu gehörigen Projekt und das Projekt zu dem Workflow.

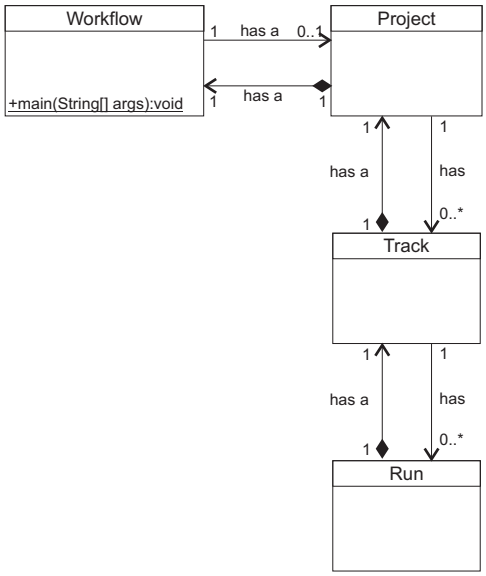


Abbildung 5.1: *Project-Track-Run* UML-Klassendiagramm I

Aus dem in Abbildung 5.1 gezeigten UML-Modell kann beispielhaft ein in Abbildung 5.2 dargestelltes Projekt mit vier Strecken und jeweils einer unterschiedlichen Anzahl von Fahrten erzeugt werden.

[2] In Together 6.0 UML-Notation.

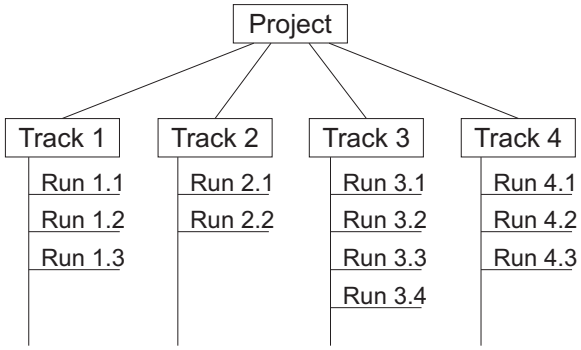


Abbildung 5.2: *Project-Track-Run* Architektur - Aufbau

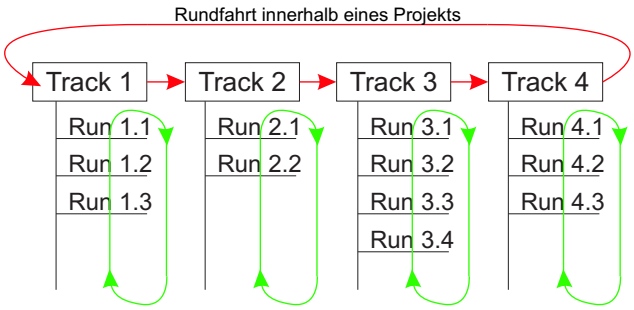


Abbildung 5.3: *Project-Track-Run* Architektur - Anwendung

Der in Abbildung 5.3 dargestellte Ablauf tritt in Verbindung mit der Projektstruktur auf. Dies beinhaltet eine benutzerdefinierte Zusammenstellung von verschiedenen Strecken (*Track1..Track4*) die nacheinander abgefahren werden können. Die Strecken werden dabei stets in der festgelegten Reihenfolge[3] abgearbeitet. Eine Strecke (*Track*) ist in eine oder mehrere Fahrten (*Run*) unterteilt, wobei während der Fahrsimulation eine beliebige Fahrt gewählt[4] werden kann.

Die Datenhaltung der Video- und GPS-Daten zur Laufzeit wird von der jeweiligen Fahrt selbst bewerkstelligt. Abbildung 5.4 zeigt das UML-Klassendiagramm[5] für die Datenverwaltung von Video- und GPS-Daten. Eine Fahrt besitzt immer einen *VideoController*, mit dem aufgenommenen Video dieser Fahrt, sowie einen *GPSContainer*, mit den dazugehörigen GPS-Daten.

[3] Rot, als *Rundfahrt innerhalb eines Projekts*, hervorgehoben.
[4] Grob schematisch in grün hervorgehoben.
[5] In Together 6.0 UML-Notation.

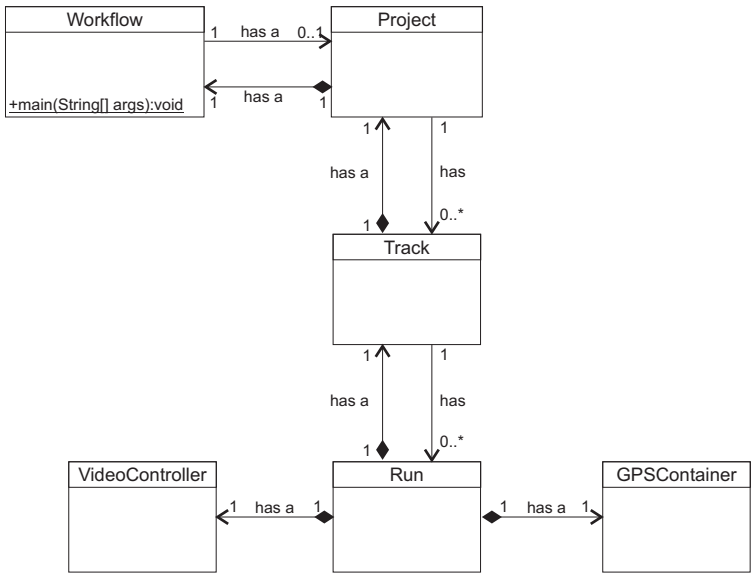


Abbildung 5.4: *Project-Track-Run* UML-Klassendiagramm II

Innerhalb eines GPSContainers werden die GPS-Daten sequentiell[6] in einem Array[7] gehalten.

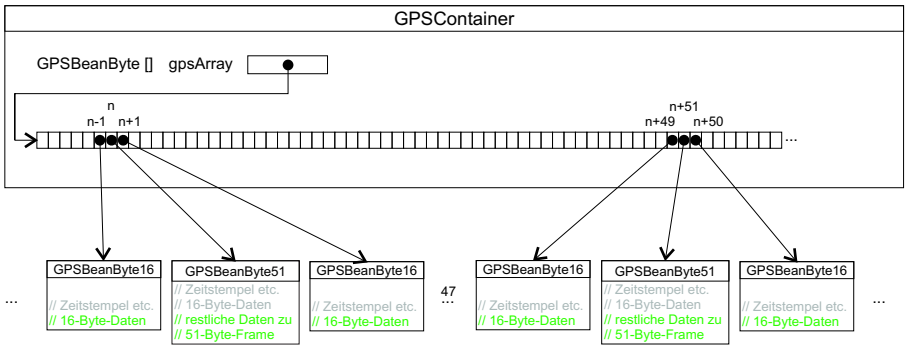


Abbildung 5.5: *GPSContainer* Architektur

Abbildung 5.5 zeigt die Datenhaltung der aufgenommenen Daten des Ortungsdatenmoduls, welche mit 50 Hz[8] aufgezeichnet wurden. Zwischen dem Auftreten zweier GPS-Positionsdaten[9] (GPSBeanByte51) sind 49 Ortungsdaten mit Gala- und Gyro-Werten[10] (GPSBeanByte16) in dem *gpsArray* enthalten.

[6] In der Reihenfolge, in der sie bei der Aufnahme des Videos aufgetreten sind. (Vgl. Abschnitt 3.3 auf Seite 18)

[7] Ein Array bietet die Möglichkeit sehr schnelle Lese- und Schreibroutinen auf den referenzierten Datenobjekten zu implementieren.

[8] Vgl. Abschnitt 4.2.2 auf Seite 24.

[9] Frequenz: 1Hz.

[10] Frequenz: 50Hz.

5.2 Problematik der räumlichen Fahrtenwechsel

Durch unterschiedliche Tages- und Jahreszeiten, sowie verschiedene Wetter- und Verkehrssituationen, die in einer Fahrt einer Strecke aufgezeichnet werden, ergeben sich verschiedene Simulationsszenarien. Steht ein Proband im Testlauf einer Simulation beispielsweise an einer Ampel, so existieren im System der Fahrsimulation mehrere verschiedene Aufnahmen dieser Position[11]. Eine Ampel-Kreuzung-Situation erscheint in einer verregneten Nacht anders als an einem sonnigen Sommertag. Unter solchen verschiedenen Szenarien ergeben sich unterschiedliche Belastungsgrade des Fahrers, die im Bereich der HMI-Forschung untersucht werden. Des Weiteren besteht auch die Möglichkeit, den Fahrer mit unerwarteten Situationen, wie sich plötzlich ändernden Fahrverhältnissen, zu konfrontieren. Dazu muss es möglich sein, an einem beliebigen Punkt der Fahrsimulation die Fahrt einer Strecke zu wechseln. Aufgrund der Eigenschaften von Video- und GPS-Daten besteht eine Differenzierung zwischen diesen beiden Datenarten. Ein Video kann als ein Fluss von (Bild-)Daten betrachtet werden[12], bei dem kontinulierlich auf jedes enthaltene Bild ein neues Bild folgt, bis das Ende des Videos erreicht ist. Ziel ist es, beim Umschalten von einer laufenden Fahrt zu einer anderen Fahrt, nicht das Video „2“ zum gleichen Zeitpunkt, sondern an der gleichen GPS-Position zu zeigen. Die Problematik dabei ist, dass dies möglichst schnell erflogen soll. Zum Anderen sind die GPS-Positionen in den aufgenommen Fahrten nahezu nie identisch.

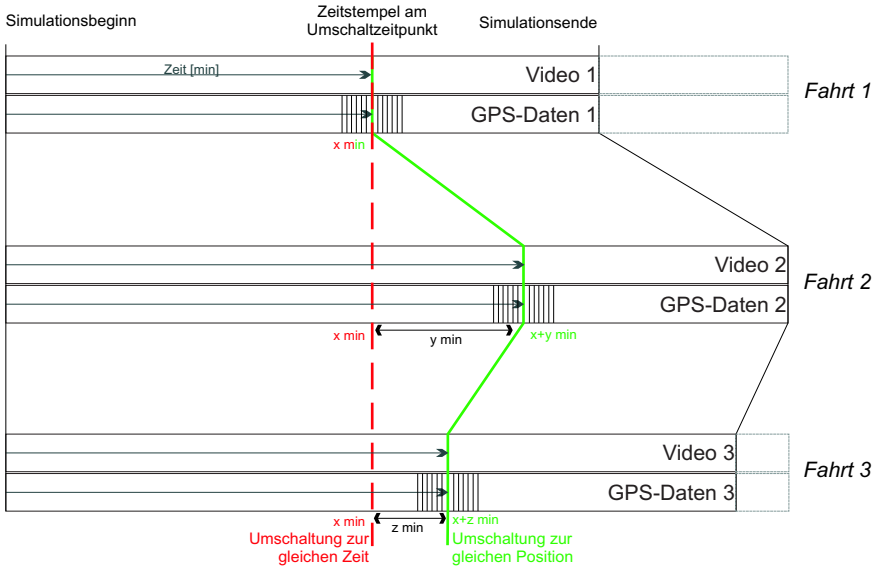


Abbildung 5.6: Umschaltung zwischen verschiedenen Fahrten einer Strecke

[11] Vgl. Abschnitt 5.1 auf Seite 35.
[12] Vgl. Abschnitt 3.1.1 auf Seite 11.

Die Abbildung 5.6 zeigt eine mögliche Differenz[13], die vom Start einer Fahrt *1* in Bezug auf die Fahrten *2* und *3* der gleichen Strecke auftreten kann. Dies kann beispielsweise durch längere Ampelaufenthalte, dichteren Verkehr oder ähnliche Vorkommnisse geschehen. Die auftretenden Zeitdifferenzen sind dabei mit *y min* (Fahrt *2*) und *z min* (Fahrt *3*) angegeben. Somit würde sich der temporale Umschaltzeitpunkt von Fahrt *1* zu der gleichen räumlichen Position in einer der anderen Fahrten unterscheiden. Ein Wechsel auf der Zeitbasis des Videos liefert demzufolge ein falsches Ergebnis. Die Umschaltung zwischen verschiedenen Fahrten während der Simulation muss also auf Basis der räumlichen Positionsdaten geschehen. In dem Abschnitt '2.2 Taxonomy of positional fusion algorithms' aus [Hall, 1992, Seite 37] heisst es: „Positional fusion algorithms provide an estimate of the state vector that best fits (in a defined mathematical sense) the observed data." Dies bedeutet sinngemäß, dass positionsabhängige Fusionsalgorithmen stets nur eine Schätzung des best möglichen Zustandsvektors als Ergebnis liefern können. Dabei wird der Zustandsvektor (wiederum nach [Hall, 1992, Seite 37]) als ein unabhängiger Satz von Variablen wie Position und Geschwindigkeit angesehen, welcher eine relativ präzise Vorhersage des zukünftigen Verhaltens einer Entität ermöglicht. Diese Aussagen münden letztendlich in der Tatsache, dass bei der Fusion von unabhängigen Positionsdatensätzen nicht immer eine genaue Zuordnung identischer Positionen erfolgt. Meist geschieht die Zuordnung bei solchen Verfahren durch Annäherungen, wobei einer gegebenen Ortsposition aus einem Positionsdatensatz eine Ortsposition eines anderen Datensatzes zugeordnet wird, die die 'nächst bessere' Position ist. Es wird also eine heuristische Schätzung vorgenommen, welche Positionen innerhalb eines Bereiches als 'zusammenpassend' betrachtet.

Um bei einer räumlichen Umschaltung zwischen den Fahrten einer Strecke[14] die richtige Stelle in einer anderen Fahrt, d.h. die gleiche oder eine sehr ähnliche GPS-Position, zu finden, werden im folgendenen Abschnitt verschiedene Lösungsansätze vorgestellt.

5.3 Lösungsansätze

5.3.1 Vollständige Suche

Die vollständige Suche beschäftigt sich mit dem Bestimmen der korrekten Position in einer anderen Fahrt in dem Moment der Anfrage. Dies erfordert einen Algorithmus, der zu einem, vom Nutzer gewünschten, Zeitpunkt die GPS-Position in der aktuellen Fahrt einer Strecke ermittelt. Diese aktuelle räumliche Position wird nach deren Bestimmung in einer anderen, vom Nutzer gewünschten, Fahrt gesucht. Dabei kann jedoch eine gewisse Ungenauigkeit auftreten, da mit sehr hoher Wahrscheinlichkeit die exakte Position in den GPS-Datensätzensätzen anderer Fahrten nicht wieder auftritt. Somit wird die 'nächst bessere'

[13] Alle derartigen Abbildungen in diesem Kapitel sind relativ. Die Abstände und Relationen folgen nicht linearen Gesetzen und sind grob schematisch dargestellt.

[14] In Abbildung 5.3 und 5.6 grün dargestellt.

Position, das heißt, die Position gewählt, die der Ausgangsposition in der ursprünglichen Fahrt am Nahesten ist.

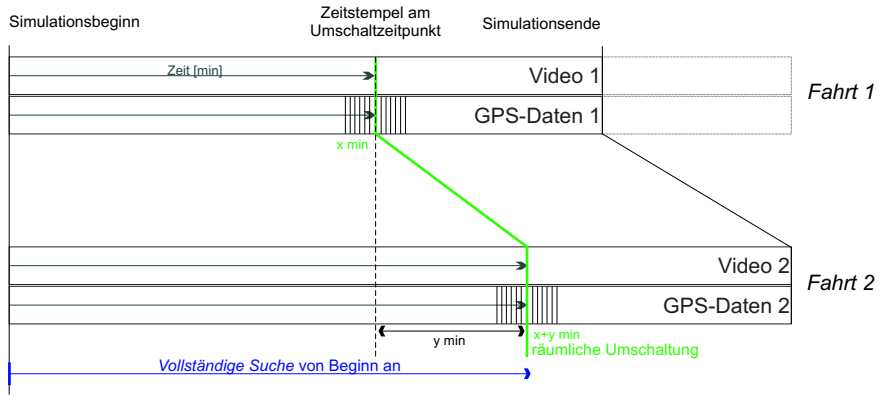


Abbildung 5.7: Vollständige Suche

Abbildung 5.7 zeigt den Vorgang der vollständigen Suche. Zu einem gewünschten Zeitpunkt in dem Video *1* wird der Umschaltvorgang aktiviert. Daraufhin wird in der gewünschten Fahrt (*2*) am Anfang begonnen und der komplette GPS-Datensatz durchsucht, bis die entsprechende (oder eine der innerhalb einer Toleranz liegende) GPS-Postition gefunden wird. Aus dieser Position wird der Zeitstempel innerhalb von Video *2* ermittelt und die Simulation wieder an diesem Punkt der anderen Fahrt (*2*) gestartet. Die Vorteile der vollständigen Suche sind schnelle und kompakte Möglichkeiten der Implementierung. Im Gegensatz dazu verursacht eine vollständige Suche der gewünschten GPS-Position zum Umschaltzeitpunkt einen hohen Aufwand an Rechenleistung.

5.3.2 Lokale Suche

Die Weiterentwicklung der vollständigen Suche ist die lokale Suche. Dabei wird davon ausgegangen, dass:

- alle GPS-Daten in der richtigen Reihenfolge, von Beginn bis Ende des Videos, zu Verfügung stehen.
- bei der Umschaltung alle Fahrten, zwischen denen umgeschalten werden kann, die Fahrten einer Strecke sind. Somit gleichen sich die GPS-Positionen innerhalb dieser Fahrten zu einem gewissen Grad.

Ist dies der Fall, kann Anhand des Zeitstempels in der Ausgangsfahrt zu der neuen Fahrt (*2*) gesprungen werden. Somit ist der Ausgangspunkt der Positionssuche in der Fahrt *2* der Zeitpunkt, an dem sich der Nutzer in der Fahrt *1* zum Umschaltzeitpunkt befindet.

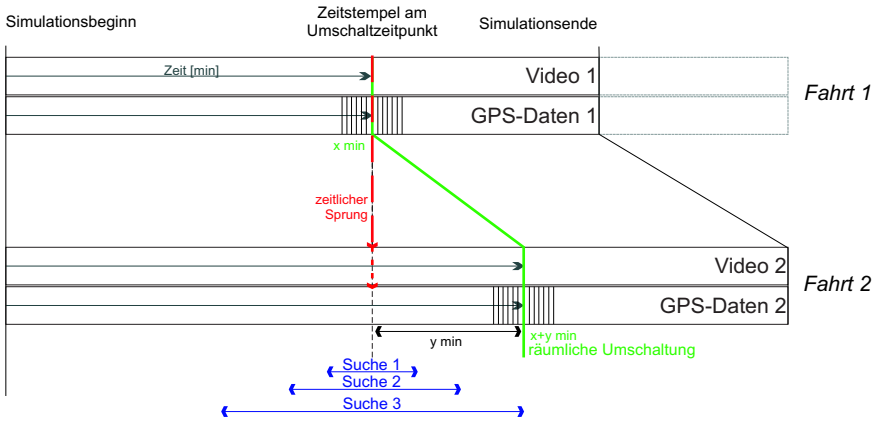


Abbildung 5.8: Lokale Suche

Abbildung 5.8 zeigt eine Bereichsuche von einer zeitlich geschätzten GPS-Position ausgehend. Anhand von definierten Toleranzen um diese GPS-Position in Fahrt *2* wird nach der gegebenen GPS-Position aus Fahrt *1* gesucht. Wird diese GPS-Position nicht gefunden, muss der Suchraum schrittweise erhöht werden bis die entsprechende GPS-Position gefunden wurde. Im Gegensatz zu der in Abschnitt 5.3.1 vorgestellten vollständigen Suche wird hierbei der Suchaufwand verringert.

5.3.3 Lokale Suche mit 'Keyframes'

Das Einfügen von Keyframes kann als eine Spezialisierung der lokalen Suche angesehen werden. Bei der lokalen Suche wird anhand des Zeitstempels direkt an die gleiche zeitliche Position in einem anderen Video gesprungen und von dort aus weitergesucht. Durch ein Einfügen von vordefinierten Keyframes in die GPS-Datensätze kann von Beginn an eine Verbindung zwischen den einzelnen Datensätzen hergestellt werden. Ein jedes dieser Keyframes hat eine Referenz zu einem (bei mehr als zwei Fahrten einer Strecke zu mehreren) korrespondierenden Keyframe in den anderen GPS-Datensätzen. Wie in Abbildung 5.9 dargestellt, treten diese Keyframes in einem bestimmten Intervall (hier *10 sec.*) auf. Aufgrund dieser vorgegebenen Frequenz kann bei der Suche einer korrespondierenden GPS-Position ziemlich genau in den richtigen Bereich gesprungen werden, von dem aus nur noch ein kleiner Suchraum betrachtet werden muss.

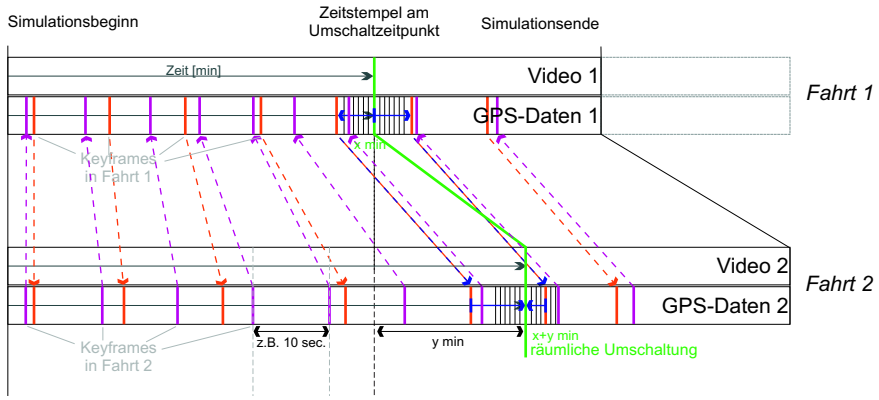


Abbildung 5.9: Beispielhafte Einbettung von Keyframes

In diesem Beispiel ist es Definitionssache, ob von dem Umschaltzeitpunkt ausgehend das nachfolgende oder das vorhergehende Keyframe als Referenz zu Fahrt *2* genommen wird. In beiden Fällen ist danach entscheidbar in welche Richtung die entsprechende Position gesucht wird. Außerdem definiert diese Vorgehensweise auch eine Obergrenze bei der Suche von dem Keyframe aus. Somit sollte die GPS-Positionssuche hierbei nicht das nächste (bzw. vorhergehende) Keyframe erreichen, da sich das gesuchte Positionsziel zwischen diesen beiden Keyframes befinden muss.

5.3.4 Indizierung aller GPS-Positionen

Eine weitere Möglichkeit der räumlichen Fusion von mehrdimensionalen GPS-Datensätzen ist die Indizierung aller GPS-Positionen, die in allen Fahrten einer Strecke auftreten. Ein Vorteil dieses Verfahrens ist die Vermeidung der Positionssuche zum Zeitpunkt des Umschaltens. Bei dem Laden der GPS- und Videodaten in das System wird ein Index erstellt, der Referenzen zwischen allen vorkommenden GPS-Positionen aller vorhandenen Fahrten enthält. An jeder Stelle in jedem Video stehen somit immer Referenzpositionen in den anderen Videos zur Verfügung.

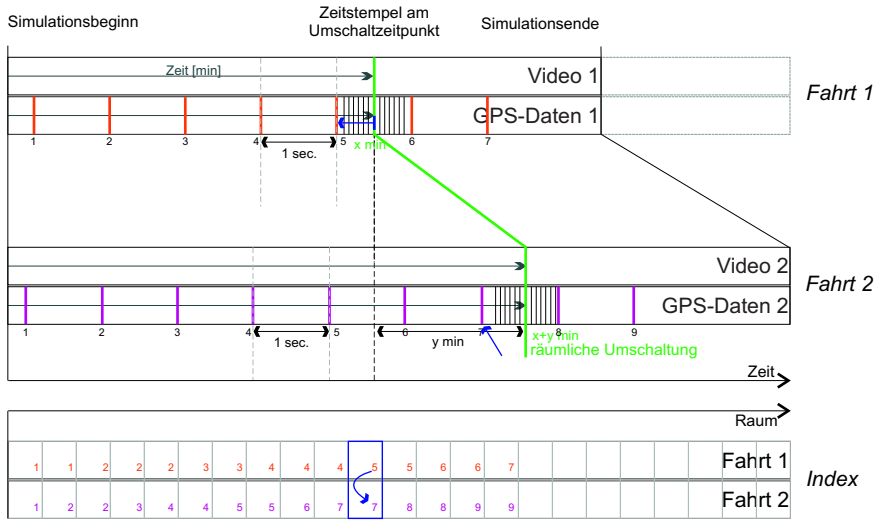


Abbildung 5.10: Indizierung von 2 Fahrten

Auf Basis des in Abbildung 5.10 dargestellten *Index* kann eine räumliche Umschaltung (grün) von Fahrt *1* zu Fahrt *2* erfolgen. Dazu wird in den GPS-Daten der Fahrt *1* die letzte gültige GPS-Position gewählt. Dessen Spalte[15] in dem *Index* referenziert die Umschaltpositionen zu den anderen Fahrten. Der *Index* besteht in diesem Beispiel aus zwei Zeilen, da es nur zwei zu indizierende Fahrten gibt. Somit gibt es für die GPS-Position *5* in Fahrt *1* (rot) eine korrespondierende GPS-Position in Fahrt *2* (Lila im unteren Bereich des blauen Kastens). In dem Beispiel-Index in Abbildung 5.10 existieren in der ersten Zeile (GPS-Positionsreferenzen für Fahrt *1*) zwei Spalten mit dem Referenzeintrag *5*. Es wird definiert, stets den ersten solcher Einträge zu wählen, da das Video *1* angehalten und das Video *2* gestartet wird. Dieser Startvorgang von Video *2* kann aufgrund interner Puffermechanismen zum Starten von Videos in dem Java Media Frameworks eine kurze Zeit in Anspruch nehmen. Somit ist es sinnvoller, eine räumlich 'frühere' GPS-Position als Startpunkt zu wählen, um diesem Puffermechanismus Zeit zu geben, das Video zu laden und zu starten.

5.4 Lösungskonzept Indizierung

Aufgrund der sehr hohen Auslastung[16] des Prozessors zur Laufzeit, insbesondere bei der Erhöhung der Abspielrate, darf eine Umschaltung zwischen verschiedenen Fahrten einer Strecke keinen zu hohen Aufwand an Rechnenleistung beanspruchen. Die vier in Abschnitt 5.3 vorgestellten Lösungsansätze müssen also in erster Linie nach diesem Kriterium be-

[15] Eine GPS-Position besitzt auch immer eine Referenz zu der korrespondierenden „Sprungspalte“ in dem Index.

[16] Vgl. Abschnitt 4.3 auf Seite 29.

trachtet werden. In den ersten drei Ansätzen muss zum Zeitpunkt des Umschaltens stets noch die richtige Position gesucht werden. Auch wenn eine höchstwahrscheinlich starke Verringerung des Suchaufwands beginnend bei der *vollständigen Suche* über die *lokale Suche*, bis hin zu der Einbettung von *'Keyframes'* vorliegt, ist die Indizierung aller GPS-Daten in allen Fahrten und dessen Verknüpfung mit dem geringsten Aufwand zur Laufzeit verbunden, da stets die Referenzen zu allen Positionen in allen Fahrten einer Strecke zur Verfügung stehen. Dabei treten jedoch konzeptionelle Fragen auf:

1. Wie werden GPS-Daten behandelt, die in mehreren Fahrten gleich sind?
2. Wie werden Haltestellen (z.B. an einer Ampel, da sich das Fahrzeug hier nicht fortbewegt) gehandhabt?
3. Wie schnell kann dieser Index berechnet werden?
4. Wie hoch ist der Speicheraufwand für den Index zur Laufzeit?

Die beiden Fragen 1 und 2 zielen in das gleiche Gebiet der Indizierung. Wenn ein Fahrzeug an einer Ampel stehen bleibt, dann lief die Aufnahme des Videos und der GPS-Daten weiter, somit werden solche Haltestellen auch in der Fahrsimulation gezeigt. In den gewonnenen GPS-Daten (mit Referenzen zu deren Stellen in dem Video) sind bestimmte fahrdynamische Parameter, wie die Geschwindigkeit, enthalten. Aus diesen Werten kann detektiert werden ob sich ein Fahrzeug in der Fahrsimulation fortbewegt oder ob es an einem räumlichen Punkt (Ampel) steht. Für die Betrachtung in dem Index sind jedoch lediglich die Positionsdaten relevant, an denen sich das Fahrzeug bewegt, sowie die Positionsdaten, an denen das Fahrzeug gerade zum Stillstand gekommen ist oder gerade anfährt. Ob die 'Anhalteposition' oder die 'Wiederanfahrtsposition' für die Berechnung des Index in betracht gezogen werden, ist Definitionssache.

In dem resultierenden Index werden, wie in Abbildung 5.10 dargestellt, alle GPS-Koordinaten mit entsprechenden Referenzen zu den GPS-Positionen in den anderen Fahrten abgebildet. Somit kann der Index höchstens so lang sein (so viele Spalten besitzen), wie es insgesamt an GPS-Positionen in allen Fahrten zusammen gibt. Diese Anzahl verringert sich noch um ein gewisses Maß, da auch fehlerhafte GPS-Daten vorkommen können und diese aussortiert werden. Des Weiteren werden die 'Haltepositionen' auch nicht mit in den Index aufgenommen, da diese für die Indexberechnung irrelevant sind. Aufgrund der Frequenz des ODM für die Bereitstellung von GPS-Koordinaten von 1 Hz, werden einmal pro Sekunde GPS-Koordinaten aufgezeichnet. Bei einer Strecke, die in einer Gesamtzeit von 10 Minuten aufgenommen wurde, würden daraus 600 GPS-Koordinaten entstehen, die bei der Indexberechnung berücksichtigt werden. Für den Fall von drei Fahrten auf dieser Strecke, entspräche dies einem Index mit max. 1800 Spalten und 3 Zeilen. Da die GPS-Daten immer in der Reihenfolge des Auftretens und synchronisiert zum Video

zur Verfügung stehen, können diese aufsteigend durchnummeriert werden. Mit Hilfe der daraus entstehenden eindeutigen ID's[17] lässt sich ein Indexeintrag auf eine einzige Zahl beschränken, die ID der zu referenzierenden GPS-Position. Somit ergibt sich ein Index von Indices, was den Speicheraufwand nicht zu groß werden lässt.

Aus einem anderen Blickwinkel betrachtet, werden alle Positionen aller Fahrten einer Strecke in ein räumliches Raster mit variablen Rasterelementen gelegt, wie in Abbildung 5.11 grob schematisch dargestellt.

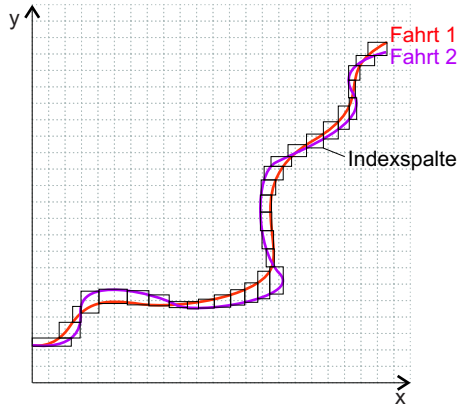


Abbildung 5.11: Fusionsrasterung von 2 Fahrten

Eine derartige Differenz bei den Positionsdaten kann durch das Fahren auf unterschiedlichen Fahrspuren, veränderten Fahrsituationen, wie einer neuen Baustelle, oder ähnlichem auftreten.

5.4.1 Mathematische Grundlagen

Der Algorithmus zur Fusion von mehrdimensionalen Video- und GPS-Daten basiert auf dem mathematischen Prinzip der senkrechten Vektorprojektion. Um die Indizierung beim Laden der Streckendaten zu vollziehen, müssen alle gültigen GPS-Positionen aller Fahrten betrachtet und in einer bestimmten Reihenfolge in den Index eingetragen werden. Die in Abbildung 5.11 dargestellten zwei Fahrten einer Strecke können prinzipiell als eine Aneinanderreihung von GPS-Positionen angesehen werden, wie in Abbildung 5.12 verdeutlicht.

[17] Diese ID ist innerhalb einer Fahrt eindeutig.

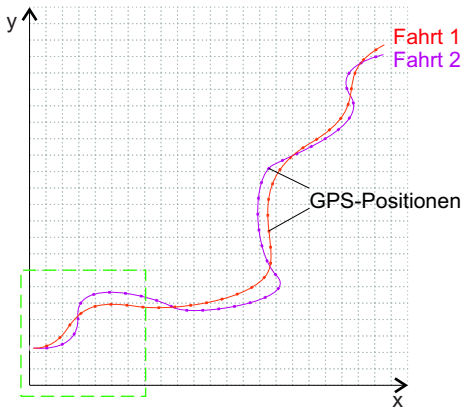


Abbildung 5.12: Aneinandergereihte GPS-Koordinaten

Der in Abbildung 5.12 dargestellte grüne Ausschnitt wird in Abbildung 5.13 noch detaillierter dargestellt. Er zeigt eine genaue Rasterung des Beispiels bei zwei Fahrten einer Strecke. In den abwechselnd blau- und schwarzgefärbten Rahmen[18] sind jeweils zwei Knoten (GPS-Positionen) enthalten, die in je eine Indexspalte eingetragen werden.

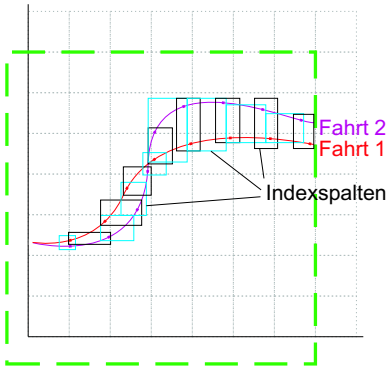


Abbildung 5.13: Vergrößert dargestellte Rasterung auf Basis von Abbildung 5.12

Um diese Paarungen in den einzelnen Rahmen zu erhalten, muss eine Abstandsberechnung auf Basis der Koordinaten vorgenommen werden. Dabei ist ein simpler Vergleich der Positionswerte und die Auswahl der geringsten Werte nicht möglich[19], da diese Punkte Koordinaten innerhalb eines Koordinatensystems repräsentieren. Ein Punkt in einem GPS-Datensatz besteht hier immer aus der geographischen Länge (x-Koordinate) und der geographischen Breite (y-Koordinate). Somit kann ein Punkt in dem Koordinatensystem auch als Vektor betrachtet werden:

[18] Der Farbwechsel soll lediglich die Übersichtlichkeit fördern und hat keine inhaltliche Bedeutung.
[19] Z.B.: Ist $(x_1 + y_1)$ kleiner, größer oder gleich $(x_2 + y_2)$?

$$\overrightarrow{GPSPosition} = \begin{pmatrix} Grad° Minute.Kommaminute \\ Grad° Minute.Kommaminute \end{pmatrix}$$

Beispiele:

$$\overrightarrow{a} = \begin{pmatrix} 9°51.220\, oestl.\, Laenge \\ 52°\ 7.380\, noerdl.\, Breite \end{pmatrix} \overrightarrow{b} = \begin{pmatrix} 9°51.200\, oestl.\, Laenge \\ 52°\ 7.400\, noerdl.\, Breite \end{pmatrix}$$

Das Bezugssystem eines solchen Vektors ist WGS84, welches die „[...] geodätische Grundlage des Global Positioning System für die Vermessung der Erde und ihrer Objekte mit Hilfe von Satelliten ist. [...]“ (Quelle: [Wiki-WGS84]) Da die unter $\overrightarrow{a}$ und $\overrightarrow{b}$ angegebenen x- und y-Koordinaten in Grad, Minute und Kommaminute eingeteilt sind, kann mit ihnen nicht ohne Weiteres eine Berechnung erfolgen. Eine solche Koordinate kann auf die Einheit Grad oder die drei Teile (Grad, Minute, Kommaminute) auf die Einheit Minute angeglichen werden. Für die Angleichung an die Einheit Minute gilt:

$$vereinheitlichte\, Koordinate = (Grad \cdot 60) + Minute + (Kommaminute \div 1000)$$

Für die Koordinaten des Vektors $\overrightarrow{a}$ würde sich somit ergeben:

$$vereinheitlichte\, xKoordinate = (9 \cdot 60) + 51 + (220 \div 1000) = 591,22$$
$$vereinheitlichte\, yKoordinate = (52 \cdot 60) + 7 + (380 \div 1000) = 3127,38$$

Daraus ergibt sich der vereinheitlichte Vektor $\overrightarrow{a}$:

$$\overrightarrow{a} = \begin{pmatrix} 591,22 \\ 3127,38 \end{pmatrix}$$

Definitionen:

Wert	Bedeutung
$\overrightarrow{a}$	GPS-Positionsvektor einer Fahrt.
$\overrightarrow{b}$	GPS-Positionsvektor einer anderen Fahrt.
$\overrightarrow{b_a}$	Ergebnisvektor der senkrechten Projektion von $\overrightarrow{b}$ auf $\overrightarrow{a}$.
$\lvert\overrightarrow{a}\rvert$	Betrag von $\overrightarrow{a}$, Länge des Vektors $\overrightarrow{a}$.
$d\left(\overrightarrow{b_a}\right)$	Distanz von $\overrightarrow{b_a}$, die relative Länge des Vektors $\overrightarrow{b_a}$ bezüglich $\overrightarrow{a}$.

Tabelle 5.1: Definitionen für das mathematische Prinzip der senkrechten Vektorprojektion

Auf Basis *vereinheitlichter Koordinaten* kann eine Weiterverarbeitung erfolgen. Um die Abstände der beiden Vektoren $\overrightarrow{a}$ und $\overrightarrow{b}$ zu dem Ursprung in Bezug auf eine gefahrene

Strecke zu berechnen, muss einer der beiden Vektoren auf den anderen projiziert werden. Dabei ist es Definitionssache, ob der Vektor $\overrightarrow{a}$ auf $\overrightarrow{b}$ oder der Vektor $\overrightarrow{b}$ auf $\overrightarrow{a}$ projiziert wird. Eine sogenannte senkrechte Vektorprojektion ist wie folgt definiert:

$$\overrightarrow{b_a} = \frac{\overrightarrow{a} \bullet \overrightarrow{b}}{|\overrightarrow{a}|^2} \cdot \overrightarrow{a}$$

Abbildung 5.14 stellt diesen Vorgang graphisch dar:

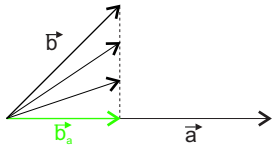


Abbildung 5.14: Senkrechte Vektorprojektion (Quelle: [StrickWurl, Seite 52])

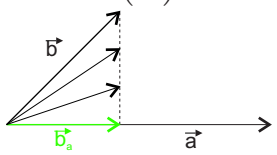
Aus dieser Projektion lässt sich die Länge des projizierten Vektors $\overrightarrow{b_a}$ in Bezug auf den Vektor $\overrightarrow{a}$ relativ zum Koordinatenursprung wie folgt berechnen:

$$d\left(\overrightarrow{b_a}\right) = \frac{\overrightarrow{a} \bullet \overrightarrow{b}}{|\overrightarrow{a}|^2} = \frac{a_1 b_1 + a_2 b_2}{a_1^2 + a_2^2}$$

Wie in Abbildung 5.14 gezeigt, kann der projizierte Vektor $\overrightarrow{b_a}$ auf der Projektion kürzer als der Projektionsvektor[20] $\overrightarrow{a}$ sein. Des Weiteren kann der Vektor $\overrightarrow{b_a}$ auch länger und gleich lang wie der Projektionsvektor $\overrightarrow{a}$ sein. Für diese drei Fälle gilt folgendes:

1. $\overrightarrow{b_a}$ ist kürzer als $\overrightarrow{a}$:

$$0 < d\left(\overrightarrow{b_a}\right) < 1$$

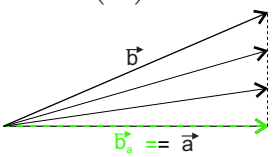


Daraus folgt: $\overrightarrow{b}$ liegt dem Ursprung am nächsten.

2. $\overrightarrow{b_a}$ ist identisch mit $\overrightarrow{a}$ (wird fast nie erreicht):

[20] Der Vektor auf den projeziert wird.

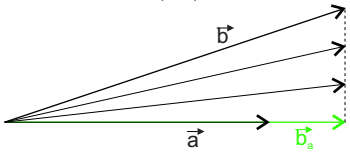
$$d\left(\overrightarrow{b_a}\right) = 1$$



Daraus folgt: Keiner der beiden Punkte ist dem Ursprung am nächsten.

3. $\overrightarrow{b_a}$ ist länger als $\overrightarrow{a}$:

$$1 < d\left(\overrightarrow{b_a}\right) < \infty$$



Daraus folgt: $\overrightarrow{a}$ liegt dem Ursprung am nächsten.

Das Ergebnis dieser Distanzberechnung kann somit für eine simple Vergleichsoperation verwandt werden. Auf Basis eines Vergleichs dieser Abstände wird entschieden, welcher der betrachteten Vektoren sich näher an dem Ursprung des zugrunde liegenden Koordinatensystems befindet. Dies ist die Grundlage für den Algorithmus zur Indexgenerierung über die senkrechte Vektorprojektion.

5.4.2 Algorithmus zur Indexgenerierung

Dieser Algorithmus durchläuft die vorhanden GPS-Datensätze aller Fahrten zeitgleich und betrachtet die GPS-Positionen immer in Bezug auf die zuletzt betrachtete GPS-Position. Da es stets ein Bezugssystem zu den zu betrachtenden GPS-Positionen geben muss, werden die GPS-Positionen immer relativ zu der zuletzt gewählten GPS-Position betrachtet.

Umfangsabschätzung des Index: Die maximale Größe eines resultierenden Index wird durch die Anzahl der vorhandenen Fahrten und der darin enthaltenen GPS-Positionen festgelegt[21]. Ein Index eines Tracks hat genau so viele Zeilen, wie Fahrten in der Strecke vorhanden sind. Die Länge des Index (Anzahl der Spalten) kann den maximalen Wert der Summe der Anzahl aller GPS-Positionen in allen Fahrten einer Strecke annehmen.
Die verschiedenen Fahrten einer Strecke sind nicht identisch aber dennoch sehr ähnlich in Bezug auf die Positionsdaten, da sich die Strecke im Allgemeinen nicht ändert. Auch die Start- und Zielpunkte der Fahrten liegen nicht exakt auf der selben Stelle, aber gleichen

[21] Vgl. Abschnitt 5.4 auf Seite 44.

sich in ihren GPS-Positionen. Die Eigenschaft der GPS-Daten, in sortierter Reihenfolge vom Start- zum Zielpunkt einer Strecke vorzuliegen, ermöglicht eine sequentielle Betrachtung der GPS-Positionen aller Fahrten in einem Iterationsschritt. Da die maximale Größe des Index vor Beginn der Indizierung fest steht, steht auch die maximale Anzahl an Iterationsschritten für die Erstellung eines Index fest. Sie entspricht genau der Anzahl der Spalten in dem resultierenden Index.

Vorgehen: Eine einmalige Iteration über alle Spalten des Index füllt diese mit den Indices der zu indizierenden Referenzen der GPS-Positionen. Bei jedem Iterationsschritt wird die aktuelle Spalte des Index gefüllt. Dies bedeutet, es werden genau so viele Indices in dieser Spalte gespeichert, wie Fahrten (Runs) in der Strecke (Track) vorhanden sind, damit zu jeder vorhandenen GPS-Position einer Fahrt stets korrespondierende GPS-Positionen zu allen anderen Fahrten vorhanden sind. Es wird in allen Datensätzen geprüft, welche der nächsten GPS-Positionen die geringste Entfernung zu der zuletzt betrachteten GPS-Position hat. Diese GPS-Position wird in das entsprechende Feld der Spalte eingetragen. Liegt die gefundene GPS-Position beispielsweise auf der Fahrt 1, so wird die Referenz[22] dieser GPS-Position an der Stelle *index[1][aktuelle_spalte]* eingetragen. Die anderen Elemente dieser Spalte werden mit den Daten aus dem vorhergehenden Iterationsschritt gefüllt. Somit ändert sich in einem Iterationsschritt lediglich ein Element der zu betrachtenden Spalte, alle anderen Elemente werden aus dem vorhergehenden Schritt übernommen.

Die Implementierung des Algorithmus' für die Indizierung der GPS-Positionen wird in Form eines Pseudo-Codes dargestellt. Vorgreifend sind folgende Variablen dafür in Tabelle 5.2 definiert.

[22] Eindeutiger Index innerhalb des dazugehörigen GPSContainers.

Definitionen:

Wert	Bedeutung
index	Der zu füllende Index
l	Laufvariable über alle Spalten des Index
u	GPS-Position, die dem aktuellen Ursprung am nahesten ist
xu,yu	Vereinheitlichte x und y Werte von u
ref(u)	Eindeutige Referenz von u in dessen GPSContainer
u2	GPS-Position, die sequentiell auf u folgt
xu2, yu2	Vereinheitlichte x und y Werte von u2
uu2	Vektor zwischen u und u2, Projektionsvektor
xuu2, yuu2	Vereinheitlichte x und y Werte von uu2
u_neu	GPS-Position, die dem Ursprung u am Nahesten ist
u_FahrtID	ID der Fahrt in der sich u befindet
anzahl_fahrten	Anzahl der Fahrten, die betrachtet werden

Tabelle 5.2: Definitionen für den Pseudo-Code der Indizierung

Pseudo-Code:

```
generateIndex():
{
  //Initialisierungen
  Spalte l=0 in index mit den Referenzen der ersten GPS-Positionen aller Fahrten füllen.
  Lege ein u als Startwert fest.
  Speichere l in u.
  for( jede Spalte l von index ){
    //Suche die Position bei u_neu, die dem aktuellen Ursprung u am nahesten ist.
    u_neu = intendNextPosition(u);
    if( u_neu ist invalid )
      break; //Der Index ist vollständig, da eine der Fahrten am Ende ist.
    u = u_neu;
    for( int i = 0; i < anzahl_fahrten; i++ ){
      Speichere l in u.
      if( i == u_FahrtID ){
        index[u_FahrtID][l] = ref(u);
        Markiere u als betrachtet.
        continue;
      }//if
      index[i][l] = index[i][l-1]; //Daten aus vorherigem Schritt übernehmen.
    }//for
    //Spalte wurde betrachtet und eine neue Position
    //wurde gefunden und im index vermerkt.
  }//for
}//generateIndex();

intendNextPosition(u):
{
  Suche in u_FahrtID nach den nächsten validen Positionsdaten
  if( u2 wurde nicht gefunden )
    return invalid u2; //Ende wurde erreicht
  Berechne x und y Werte für Projektionsvektor
  xuu2 = xu - xu2;
  yuu2 = yu - yu2;
  Berechne Distanz von u2 zu u
  Projiziere alle Positionen der restlichen Fahrten auf uu2
   und berechne deren Distanzen relativ zu uu2
  u_neu = Position mit der kleinsten der berechneten Distanzen
  return u_neu;
}//intendNextPosition();
```

Beispiel:

Im Folgenden wird dieser Vorgang anhand von drei Fahrten a (rot), b (grün) und c (blau) von Beginn an beispielhaft durchgeführt. Die einzelnen GPS-Positionen auf den jewei-

ligen Fahrten werden durch die Punkte mit sequentiell durchnummerierten Indices gekennzeichnet. Abbildung 5.15 zeigt die drei Fahrten a, b, c und deren aufgenommene GPS-Positionen zu Beginn der jeweiligen Fahrten. Außerdem sind nicht nur die Vektoren zu den Punkten ($\overrightarrow{a_1}$, $\overrightarrow{a_2}$, $\overrightarrow{b_1}$, $\overrightarrow{b_2}$, $\overrightarrow{c_1}$, $\overrightarrow{c_2}$ und $\overrightarrow{c_3}$) innerhalb des WGS84-Systems dargestellt, sondern auch die Zwischenvektoren zwischen diesen Punkten ($\overrightarrow{a_{12}}$, $\overrightarrow{a_{23}}$, $\overrightarrow{b_{12}}$,$\overrightarrow{b_{23}}$, $\overrightarrow{c_{12}}$ und $\overrightarrow{c_{23}}$) innerhalb der jeweiligen Strecke.

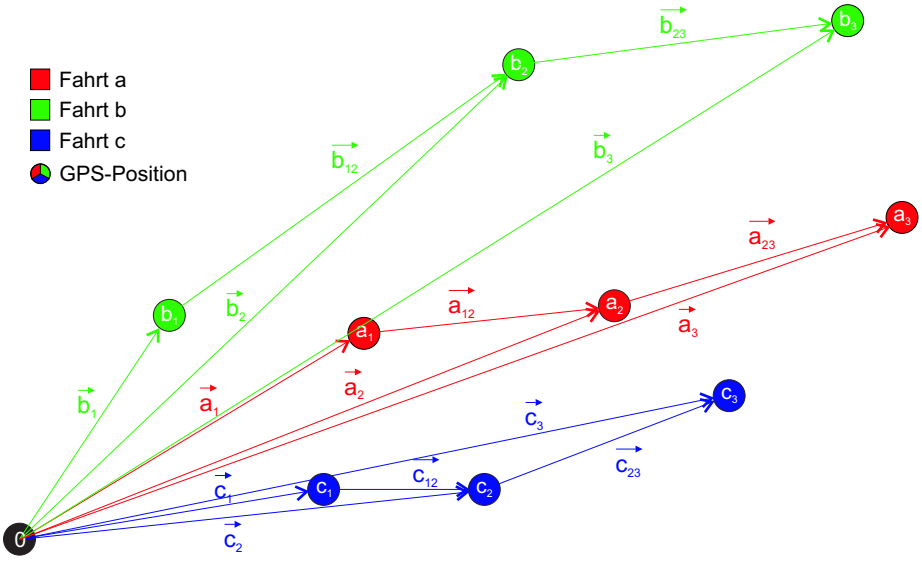


Abbildung 5.15: Beispiel - Drei Fahrten einer Strecke

Die in Abbildung 5.15 dargestellten Vektoren ergeben sich aus der Aufzeichnung der drei unterschiedlichen Fahrten. Die zuvor angesprochenen Zwischenvektoren wurden bei einer Fahrt tatsächlich abgefahren, wohingegen die Vektoren mit dem Bezug zum Ursprung des WGS84-Systems aufgezeichnet wurden. Für eine Betrachtung bei der senkrechten Vektorprojektion sind also diese Zwischenvektoren (z.B. $\overrightarrow{b_{12}}$) ausschlaggebend, die aus den Punktvektoren (z.B. $\overrightarrow{b_1}$ und $\overrightarrow{b_2}$) im WGS84-System berechnet werden können. Somit ergibt sich die in Abbildung 5.16 dargestellten Vektoren für die tatsächlichen Routen der jeweiligen Fahrten.

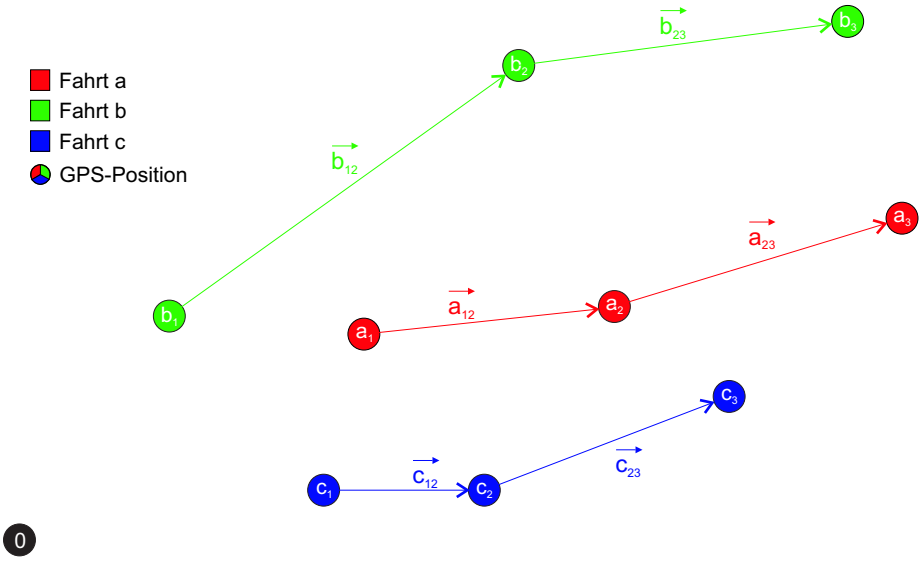


Abbildung 5.16: Beispiel - Drei Fahrten einer Strecke - Zwischenvektoren

Im ersten Iterationsschritt werden die Punkte a_1, b_1 und c_1 in Bezug auf den Koordinatenursprung (Position (0,0) des World Geodetic System 84) betrachtet. Auf Basis dieser Festlegung wird ein Einstiegspunkt innerhalb der Fahrten gefunden, anhand dessen der nächste Iterationsschritt eingeleitet werden kann. Durch die senkrechte Vektorprojektion wird entschieden, welcher der Punkte a_1, b_1 oder c_1 den größten Abstand zu dem Ursprung relativ zu dem Vektor $\overrightarrow{a_1}$ besitzt. In diesem ersten Iterationsschritt ist auch definiert, dass alle Projektionen auf den Vektor $\overrightarrow{a_1}$ des ersten Punktes a_1 der Fahrt a projiziert werden.

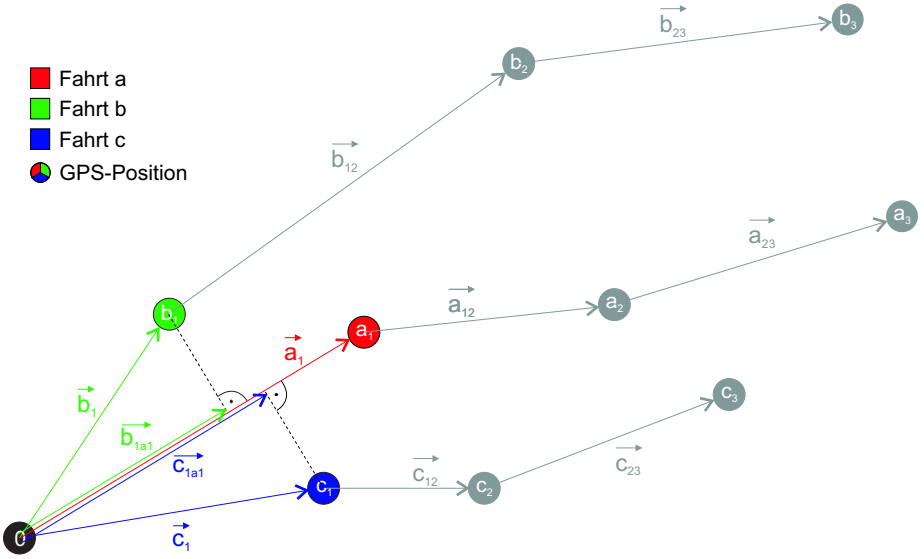


Abbildung 5.17: Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 1

Aus den in Abbildung 5.17 dargestellten Projektionen sind die beiden Vektoren $\overrightarrow{b_{1a1}}$ und $\overrightarrow{c_{1a1}}$ durch senkrechte Vektorprojektionen entstanden. Eine Maximalwertfunktion über die Distanzen der drei Vektoren $\overrightarrow{a_1}$, $\overrightarrow{b_{1a1}}$ und $\overrightarrow{c_{1a1}}$ liefert somit den Vektor, der am längsten in Bezug zu dem Vektor $\overrightarrow{a_1}$ ist.

$$MAX(\overrightarrow{a_1}, \overrightarrow{b_{1a1}}, \overrightarrow{c_{1a1}}) \Rightarrow \overrightarrow{a_1}$$

Daraus folgt: $\overrightarrow{a_1}$ hat den größten Abstand zum Ursprung 0.

Der Punkt a_1 ist die entfernteste Position ausgehend vom Ursprung 0.

Wie in Abbildung 5.18 dargestellt werden alle Indices der Startpositionen aller Fahrten zur Initialisierung in die Spalte 1 (Nummer des Iterationsschrittes) des *Index* eingetragen. Des Weiteren werden diese Positionen als 'betrachtet' markiert.

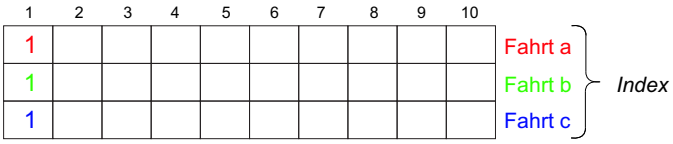


Abbildung 5.18: Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 1 *Index*

Da der Punkt a_1 in dem ersten Iterationsschritt den größten Abstand zu seinem Ursprung hat, wird der Ursprung des Koordinatensystems für den nächsten Iterationsschritt auf diesen Punkt gelegt. Dies bedeutet, dass alle Berechnungen in dem folgenden Berechnungsschritt relativ zu dem Punkt a_1 und dessen Vektor $\overrightarrow{a_1}$ erfolgen. Alle anderen Positionen, die im räumlichen Kontext in allen Fahrten vor der Position bei a_1 liegen werden nicht weiter betrachtet, sie verweisen jedoch alle bei einer Umschaltung auf die erste Spalte des Index' und haben somit auch immer eine Referenz zu den korrespondierenden GPS-Positionen in den anderen Fahrten.

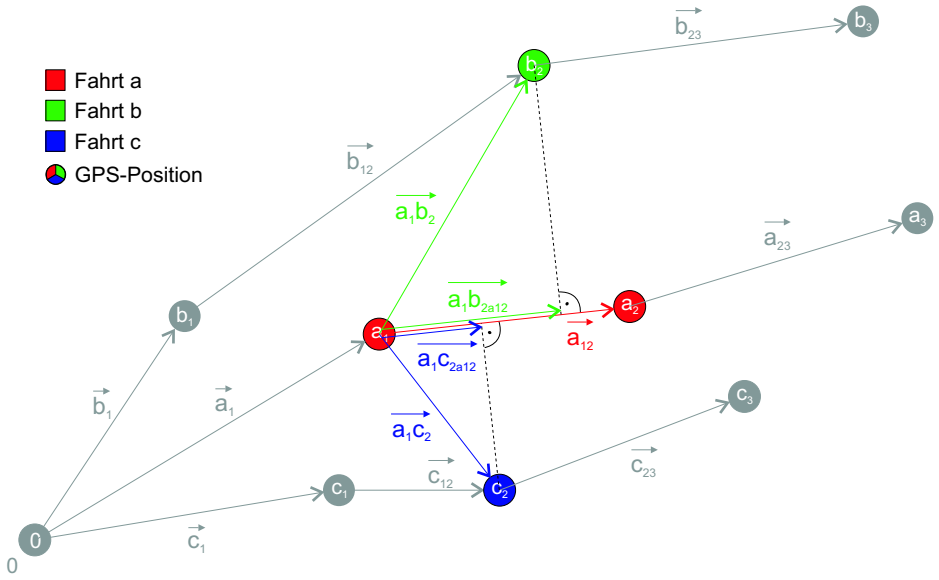


Abbildung 5.19: Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 2

In dem zweiten Iterationsschritt auf Abbildung 5.19 wird der im ersten Iterationsschritt gewählte Punkt a_1 als neuer Ursprung im Koordinatensystem angesehen. Von ihm ausgehend wird die nächste GPS-Position innerhalb der Fahrt a (a_2) gesucht. Alle noch nicht betrachteten GPS-Positionen der anderen Fahrten (b_2 und c_2) werden relativ zu dem Punkt a_1 bewertet. Somit ergeben sich zwei neue Zwischenvektoren $\overrightarrow{a_1b_2}$ und $\overrightarrow{a_1c_2}$. Diese beiden neu entstandenen Zwischenvektoren werden auf den Zwischenvektor $\overrightarrow{a_{12}}$ projiziert, und es ergeben sich die projizierten Vektoren $\overrightarrow{a_1b_{2a12}}$ und $\overrightarrow{a_1c_{2a12}}$. Von diesen beiden Ergebnisvektoren sowie dem Projektionsvektor $\overrightarrow{a_{12}}$ wird die geringste Distanz relativ zu $\overrightarrow{a_{12}}$ ermittelt.

$$MIN(\overrightarrow{a_{12}}, \overrightarrow{a_1b_{2a12}}, \overrightarrow{a_1c_{2a12}}) \Rightarrow \overrightarrow{a_1c_{2a12}}$$

Daraus folgt: $\overrightarrow{a_1c_{2a12}}$ hat den geringsten Abstand zum Ursprung a_2.

Der Punkt c_2 befindet sich dem Ursprung a_2 am Nächsten und wird übernommen.

Aus diesen Erkenntnissen ergeben sich die Einträge in der zweiten Spalte im *Index*, wie in Abbildung 5.20 dargestellt. Der Index des Punktes c_2 wird somit in dem Feld *index[Fahrt c][2]* eingetragen. Die fehlenden Werte dieser Spalte *2* werden aus dem vorhergehenden Schritt übernommen.

1	2	3	4	5	6	7	8	9	10	
1	1									Fahrt a
1	1									Fahrt b
1	2									Fahrt c

Index

Abbildung 5.20: Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 2 *Index*

Da der Punkt c_2 in dem zweiten Iterationsschritt den geringsten Abstand zu seinem Ursprung hat, wird der Ursprung des Koordinatensystems für den nächsten Iterationsschritt auf diesen Punkt gelegt. Dies bedeutet, dass alle Berechnungen in dem folgenden Berechnungsschritt relativ zu dem Punkt c_2 und dessen Vektor $\overrightarrow{c_2}$ erfolgen.

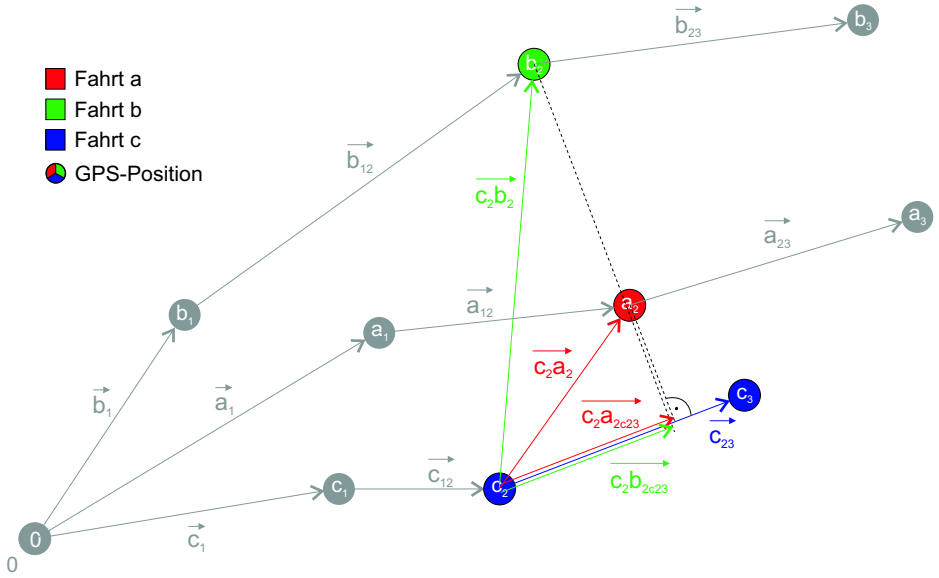


Abbildung 5.21: Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 3

In dem dritten Iterationsschritt auf Abbildung 5.21 wird der im zweiten Iterationsschritt gewählte Punkt c_2 als neuer Ursprung im Koordinatensystem angesehen. Von ihm ausgehend wird die nächste GPS-Position innerhalb der Fahrt c (c_3) gesucht. Alle noch nicht betrachteten GPS-Positionen der anderen Fahrten (b_2 und a_2) werden relativ zu dem Punkt c_2 bewertet. Somit ergeben sich zwei neue Zwischenvektoren $\overrightarrow{c_2a_2}$ und $\overrightarrow{c_2b_2}$. Diese beiden neu entstandenen Zwischenvektoren werden auf den Zwischenvektor $\overrightarrow{c_{23}}$ projiziert, und es ergeben sich die projizierten Vektoren $\overrightarrow{c_2a_{2c23}}$ und $\overrightarrow{c_2b_{2c23}}$. Von diesen beiden Ergebnisvektoren sowie dem Projektionsvektor $\overrightarrow{c_{23}}$ wird die geringste Distanz relativ zu $\overrightarrow{c_{23}}$ ermittelt.

$$MIN(\overrightarrow{c_{23}}, \overrightarrow{c_2a_{2c23}}, \overrightarrow{c_2b_{2c23}}) \Rightarrow \overrightarrow{c_2b_{2c23}}$$

Daraus folgt: $\overrightarrow{c_2b_{2c23}}$ hat den geringsten Abstand zum Ursprung c_2.

Der Punkt b_2 befindet sich dem Ursprung c_2 am Nächsten und wird übernommen.

Aus diesen Erkenntnissen ergeben sich die Einträge in der dritten Spalte im *Index*, wie in Abbildung 5.22 dargestellt. Der Index des Punktes b_2 wird somit in dem Feld *index[Fahrt b][3]* eingetragen. Die fehlenden Werte dieser Spalte *3* werden aus dem vorhergehenden Schritt übernommen.

1	2	3	4	5	6	7	8	9	10	
1	1	1								Fahrt a
1	1	2								Fahrt b
1	2	2								Fahrt c

Index

Abbildung 5.22: Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 3 *Index*

Da der Punkt b_2 in diesem Iterationsschritt den geringsten Abstand zu seinem Ursprung hat, wird der Ursprung des Koordinatensystems für den nächsten Iterationsschritt auf diesen Punkt gelegt. Dies bedeutet, dass alle Berechnungen in dem folgenden Berechnungsschritt relativ zu dem Punkt b_2 erfolgen.

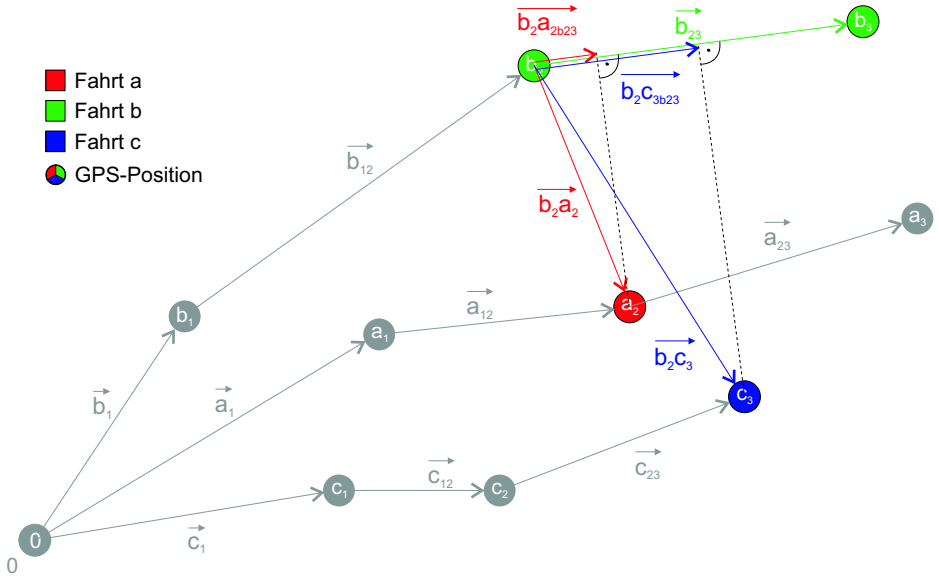


Abbildung 5.23: Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 4

In dem vierten Iterationsschritt auf Abbildung 5.23 wird der im dritten Iterationsschritt gewählte Punkt b_2 als neuer Ursprung im Koordinatensystem angesehen. Von ihm ausgehend wird die nächste GPS-Position innerhalb der Fahrt b (b_3) gesucht. Alle noch nicht betrachteten GPS-Positionen der anderen Fahrten (a_2 und c_3) werden relativ zu dem Punkt b_2 bewertet. Somit ergeben sich zwei neue Zwischenvektoren $\overrightarrow{b_2a_2}$ und $\overrightarrow{b_2c_3}$. Diese beiden neu entstandenen Zwischenvektoren werden auf den Zwischenvektor $\overrightarrow{b_{23}}$ projiziert, und es ergeben sich die projizierten Vektoren $\overrightarrow{b_2a_{2b23}}$ und $\overrightarrow{b_2c_{3b23}}$. Von diesen beiden Ergebnisvektoren sowie dem Projektionsvektor $\overrightarrow{b_{23}}$ wird die geringste Distanz relativ zu $\overrightarrow{b_{23}}$ ermittelt.

$$MIN(\overrightarrow{b_{23}}, \overrightarrow{b_2a_{2b23}}, \overrightarrow{b_2c_{3b23}}) \Rightarrow \overrightarrow{b_2a_{2b23}}$$

Daraus folgt: $\overrightarrow{b_2a_{2b23}}$ hat den geringsten Abstand zum Ursprung b_2.

Der Punkt a_2 befindet sich dem Ursprung b_2 am Nächsten und wird übernommen.

Aus diesen Erkenntnissen ergeben sich die Einträge in der vierten Spalte im *Index*, wie in Abbildung 5.24 dargestellt. Der Index des Punktes a_2 wird somit in dem Feld *index[Fahrt a][4]* eingetragen. Die fehlenden Werte dieser Spalte *4* werden aus dem vorhergehenden Schritt übernommen.

1	2	3	4	5	6	7	8	9	10	
1	1	1	2							Fahrt a
1	1	2	2							Fahrt b
1	2	2	2							Fahrt c

Index

Abbildung 5.24: Beispiel - Drei Fahrten einer Strecke - Iterationsschritt 4 *Index*

Da der Punkt a_2 in diesem Iterationsschritt den geringsten Abstand zu seinem Ursprung hat, wird der Ursprung des Koordinatensystems für den nächsten Iterationsschritt auf diesen Punkt gelegt. Dies bedeutet, dass alle Berechnungen in dem folgenden Berechnungsschritt relativ zu dem Punkt a_2 erfolgen. Diese beispielhaften Iterationsschritte sollen die Vorgehenweise des Algorithmus' zur Fusion mehrdimensionaler Video- und GPS-Daten über die Methode der senkrechten Vektorprojektion der GPS-Positionen verdeutlichen. Aufgrund der Tatsache, dass die GPS-Daten Referenzen zu ihren Positionen in dem korrespondierenden Video enthalten, müssen nur die GPS-Positionen zur Fusion dieser Daten betrachtet werden. Die Videodaten werden also vorerst nicht betrachtet, da stets eine eindeutige Abbildung einer GPS-Position auf die entsprechende Stelle in dessen Video möglich ist. Dennoch besteht stets eine Verbindung zwischen den Video-

und GPS-Daten. Somit ergibt eine räumliche Fusion der GPS-Daten auch eine Fusion der Video-Daten, wobei die Basis für die Fusionierung die Raumkoordinaten sind.

5.4.3 Bewertung

Die Indizierung aller GPS-Positionen einer Strecke über deren Fahrten minimiert den Rechenaufwand zum Zeitpunkt der Umschaltung erheblich. Im Vergleich zu der vollständigen, der lokalen Suche sowie der lokalen Suche mit Keyframes steht hierbei stets eine 'Sprungtabelle' (*Index*) zur Verfügung. Anhand derer genau festgelegt ist, welche GPS-Position in welcher Fahrt mit anderen GPS-Positionen in anderen Fahrten korreliert. Dieser *Index* erlaubt ein performantes räumliches Umschalten zwischen verschieden Fahrten einer Strecke. Dabei gibt es jedoch spezielle Rahmenbedingen, die durch die Eigenschaft einer *Fahrsimulation basierend auf realen Video- und GPS-Daten* beruhen. Ein Video bringt stets eine Vielzahl von Daten mit sich, die bei einer Umschaltung zwischen zwei Fahrten verarbeitet werden müssen. Dies bedeutet: Die Ausgangsfahrt muss ihren Verarbeitungsprozess stoppen, die neue Position in der zu startenden Fahrt muss im *Index* nachgeschlagen werden. Die neue Fahrt muss außerdem auf diese Startposition gesetzt und gestartet werden. Vor allem dieser Startvorgang des neuen Videos ist mit einer kurzen Verzögerung verbunden, da das JMF stets die Videodaten erst zwischenpuffert bevor diese wiedergegeben werden. Des Weiteren erhöht der hier verwendete MPEG4-CoDec XviD die Verzögerung auch noch einmal um ein gewisses Maß. Da dieser CoDec auf Basis von Keyframes arbeitet und zwischen diesen Keyframes nur die Veränderung in dem Bild speichert, muss erst ein entsprechendes Keyframe gefunden und dessen Daten ausgewertet werden. Nichtsdestotrotz arbeitet diese Methode zuverlässig und stellt die gewünschte Funktionalität unter den gegebenen Rahmenbedingungen zur Verfügung.

6 Fazit

6.1 Fahrsimulation im Kontext von HMI

Im Rahmen dieser Diplomarbeit wurden verschiedene Softwarekonzepte im Hinblick auf die Erstellung einer *videobasierten Fahrsimulation basierend auf realen Video- und GPS-Daten* vorgestellt. Auf der Grundlage dieser Konzepte sind zwei Softwarekomponenten entstanden. Ein Aufnahmemodul (GPS- & Video-Recording-Module), welches in der Lage ist, Video- und GPS-Daten während einer Autofahrt aufzuzeichnen. Dabei werden die Video- und GPS-Daten separat, aber dennoch in einem synchronisierten Format, abgespeichert. Dieses Format stellt die Möglichkeit zur Verfügung, die gewonnenen Daten in einer anderen Anwendung synchronisiert weiter zu verarbeiten. Auf Basis dieser vorhandenen Daten wurde ein Simulationsmodul (*VideoSim*) erstellt, welches die aufgezeichneten Video- und GPS-Daten verarbeitet und synchronisiert wiedergibt. Die Videodaten können auf einem Bildschirm ausgegeben werden. Gleichzeitig besteht die Möglichkeit, die korrelierenden GPS-Daten des aktuell ausgegebenen Videobildes an Navigationssysteme oder ähnliche GPS-verarbeitende Geräte zu liefern. Des Weiteren ist eine interaktive Beeinflussung der Fahrsimulation, beispielsweise durch eine Geschwindigkeitsänderung, möglich. Hierdurch wird dem Fahrer das Gefühl gegeben, er befände sich in einem Fahrzeug und würde den aufgezeichneten Weg entlang fahren. Eine weitere Besonderheit ist dabei die Möglichkeit des Umschaltens auf Basis der räumlichen Koordinaten zu einer anderen Fahrt durch den Fahrer oder einen anderen Benutzer der Fahrsimulation. Dies bietet den Vorteil den Fahrer mit einer unbekannten Fahrsituation zu konfrontieren. Für die HMI-Forschung bei Bosch bietet es eine Möglichkeit zur Untersuchung der Belastungsgrade von Autofahrern und inwieweit die Autofahrer von Fahrerassistenzsystemen (wie beispielsweise einem Navigationssystem) in unterschiedlichen Situationen abgelenkt werden. Dadurch sind Versuche mit unterschiedlichen Probanden mit absolut identischen Versuchsparametern reproduzierbar. Im Vergleich zum 'normalen' Fahren in einem PKW, wo die Reproduzierbarkeit nicht unbedingt gegeben ist, können somit aussagekräftige Versuchsreihen durchgeführt und ausgewertet werden. Ausgehend von der implementierten Projektstruktur in der Fahrsimulation ist außerdem die Erstellung von unterschiedlichen Simulationsszenarien mit sich unterscheidenden Strecken und Fahrten möglich. Für das Aufnahme- und das Simulationsmodul kann jeder beliebige Computer mit einem Windows

Betriebssystem genutzt werden, was die Flexibilität in der Anwendung erhöht. Es wird keine Sonderausstattung im Bereich der Hardware benötigt.

Aus den, in dieser Diplomarbeit, vorgestellten Konzepten wurde eine Fahrsimulation (*VideoSim*) entwickelt, die die dargestellten Problemstellungen bewältigt und zum Einsatz in der HMI-Forschung bei Bosch zur Verfügung steht. Abbildung 6.1 zeigt einen Screenshot dieser Fahrsimulation mit allen seinen Dialogen. Diese Dialoge beinhalten ein *Kontrollfenster*, mit dem die Fahrsimulation gestartet und gestoppt (dadurch wird die *Simulationsanzeige* aktiviert) werden kann. Von diesem *Kontrollfenster* ausgehend ist es möglich während der Fahrt die aktuellen GPS-Daten in der *GPS-Datenanzeige* anzuzeigen. Außerdem kann die Anbindung an Navigationssysteme über den Reiter *Connections* gesteuert werden. Des Weiteren ist es möglich, durch dieses *Kontrollfenster* den *Projektdialog* anzuzeigen, der es erlaubt, Projekte zu erstellen, zu speichern, zu laden und zu editieren.

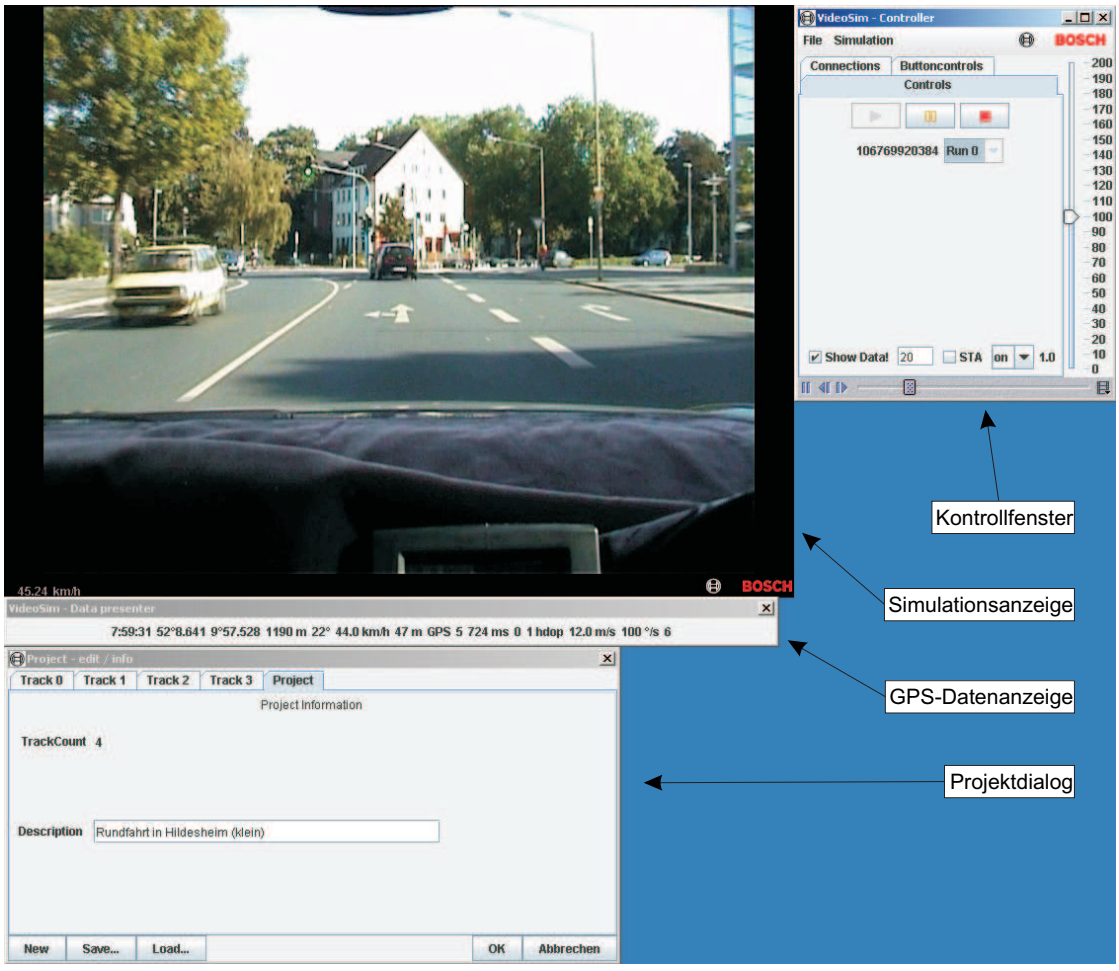


Abbildung 6.1: Screenshot *VideoSim*

Abbildung 6.2 zeigt die entwickelte Fahrsimulationssoftware nach der Installation im Testlabor für die HMI-Forschung bei der Abteilung FV/SLH der Robert Bosch GmbH in Hildesheim. An der Mittelkonsole ist ein Display angebracht, auf dem zukünftige Navigationssysteme dargestellt werden. Durch die Integration des Fahrsimulators *VideoSim* erhalten diese Navigationssysteme GPS-Daten von dem Simulator und geben anhand dieser Daten Routenempfehlungen aus.

Abbildung 6.2: *VideoSim* im Fahrsimulator zur HMI-Forschung

6.2 Perspektiven

Ausgehend vom aktuellen Standpunkt der Erkenntnisse und der realisierten Softwarekomponenten ergibt sich eine Vielzahl von Perspektiven, die aufbauend auf dieser Basis untersucht werden können.

Nicht nur das Aufzeichnen der Fahrerperspektive bei Betrachtung durch die Frontscheibe des Fahrzeugs, sondern auch der Außen- und Innenspiegel wäre eine sinnvolle Ergänzung. Hierdurch besteht die Möglichkeit, dem Fahrer der Fahrsimulation weitere Informationsquellen zur Verfügung zu stellen. Darüber hinaus liefert das Fahrzeug über den fahrzeuginternen CAN-Bus eine Vielzahl von Daten, die bei einer Aufnahmefahrt zusätzlich aufgezeichnet werden können. Dabei handelt es sich beispielweise um Informationen

über den Betrieb von sicherheitsrelevanten Geräten, wie dem Scheibenwischer, die Scheinwerfer, die Lichtblinkanlage sowie der Daten verschiedenster, im Fahrzeug vorhandener, Sensoren. In einem erweiterten Simulationssystem könnten diese Daten ausgewertet und durchaus auch manipuliert werden um bestimmte fahrsituationsbezogene Belastungsgrenzen des Fahrers zu erreichen und zu untersuchen. Die Simulation würde sich der Realität noch mehr annähern und somit kommende Versuchsreihen und deren Ergebnisse noch aussagekräftiger gestalten.

Eine weitere Möglichkeit bei der Erweiterung der Fahrsimulation wäre eine Verknüpfung der Video- und GPS-Daten mit digital kartographierten Daten. Da solche Daten aus einzelnen Wegpunkten (Kreuzungen) und Kanten (Verbindungen zwischen Kreuzungen) bestehen, könnte das Aufnahmemodul so weit umgebaut werden, dass es erkennt, wenn eine neue Kante befahren wird und somit eine neue Video- und eine neue GPS-Datei anlegt. Die Verwaltung dieser Daten in einer Datenbank für einen bestimmten Bereich (beispielsweise ein Stadtgebiet) bietet auch sehr viele Vorteile. Da jede aufgezeichnete Kante Video- sowie GPS-Daten zu dieser Fahrt besitzt, können diese Kanten beliebig aneinander gereiht werden und die Flexibilität der Fahrsimulation erhöhen. Ein Abbiegevorgang an einer Kreuzung wäre dem Fahrer der Simulation überlassen. Er könnte sich somit entscheiden, welche Straße er entlang fahren möchte. Des Weiteren könnten die aufgezeichneten GPS-Daten, welche einer gewissen Fehlerbehaftung unterliegen, auf dieses digitale Kartennetz angepasst werden und somit die Genauigkeit in den angeschlossenen Navigationssystemen erhöhen. Auch das Einfügen und Verarbeiten von verschiedenen kantenbezogenen Informationen würde diese Art des Datenmanagements erleichtern. Eine Datenbank mit verschiedenen Audiosamples wie Motorgeräuschen oder ähnlichem wäre denkbar. Diese könnten anhand der Fahrcharakteristika beispielsweise eine Motorengeräuschänderung beim Beschleunigen oder Bremsen innerhalb der Simulation hervorrufen, welches die Fahrsimulation wiederum etwas weiter in die Richtung des realen Autoverkehrs bewegt.

Auch eine Client-Server-Architektur für die Verwaltung und Verarbeitung derartiger Video- und GPS-Daten wäre denkbar. Dabei würde der Server das entsprechende Streckennetz mit den Kanten und deren Video- und GPS-Daten verwalten und bei Bedarf zur Verfügung stellen. Des Weiteren könnte er die synchronisierte Verarbeitung dieser Daten bewerkstelligen, also das Senden des Videos über ein Netzwerk an einen anfragenden Client (beispielsweise über die RTP/RTSP/RTCP Protokollfamilie) und der GPS-Daten an ein Navigationssystem. Der empfangende Client des Videostroms hätte lediglich die Aufgabe, die Videodaten darzustellen und die interaktive Beeinflussung der Simulation (beispielsweise das Beschleunigen) an den Server weiter zu reichen. Dabei müsste allerdings untersucht werden, inwieweit eine solche Architektur noch 'interaktiv' ist, oder ob es nicht vertretbare Verzögerungen in diesem Bereich gibt.

Zusätzlich besteht auch die Möglich auf Basis dieser Fahrsimulation ein Modul zu entwickeln, welches sogenannte „Points of Interest“ oder auch „Landmarks“, also wichtige Stellen an der Fahrtstrecke (wie Ampeln, Kirchen, etc.), betrachtet. Liegen beispielsweise derartige markante Stellen auf der Route des Fahrers, könnten diese in die Navigationsempfehlungen für den Fahrer mit einfließen. Navigationshinweise anhand dieser „Points of Interest“ eröffnen ganz neue Wege. Beispielsweise würde ein aktuelles, handelsübliches Navigationssystem dem Fahrer an einer bestimmten Stelle mitteilen: „In einhundert Metern bitte links abbiegen.“. Auf Basis dieser „Points of Interest“ wäre es möglich, dass das Navigationssystem den Fahrer mit dem Hinweis „Bitte links an der Kirche abbiegen.“ zur Fahrtrichtungsänderung bewegt. Somit wäre dies keine Erweiterung des Moduls *VideoSim* selbst, sondern eher aufsetzend auf den bestehenden Kommunikationsschnittstellen. Also würde ein solches System beispielsweise innerhalb angeschlossener, GPS-Daten verarbeitender, Geräte (wie einem Navigationssystem) arbeiten.

Nichtsdestotrotz bietet die aktuelle Softwarebasis der beiden Module eine Vielzahl an Perspektiven, die in Zukunft untersucht werden können. Diese beiden Module ermöglichen einen großen Spielraum an Ansatzpunkten. Aufgrund ihrer flexiblen und modularen Struktur in der Eigenschaft ihrer Implementierung sind sie mit relativ geringem Aufwand erweiterbar.

Literaturverzeichnis

[Hall] Hall, David L.: „Mathmatical Techniques in Multisensor Data Fusion“, Artech House Inc., 1992

[Antony] Antony, Richard T.: „Principles of Data Fusion Automation“

[Kynast] Kynast, Andreas: „HMI im Fahrzeug - Intelligente Bedienassistenzkonzepte“, Intranet FV/SLH Robert Bosch GmbH, 2002

[StrickWurl] Strick & Wurl: „Formelsammlung, Mathematik“, Schroedel Schulbuchverlag, 1991

[Piechulla] Piechulla, Walter: „Eine einfache Versuchsanordnung zur Simulation des Autofahrens“, Aufsatz
URL [Juli 2004] `http://www.walterpiechulla.de/SimulatorAufsatz.pdf`

[UML] Hochschule Harz: „Projekt Together“, UML Referenz
URL [Juni 2004] `http://www.projekt-together.de/vortrag/vortrag.htm`

[Wiki-OSGi] de.wikipedia.org: „OSGi“, Wikipedia - Die freie Enzyklopädie
URL [August 2004] `http://de.wikipedia.org/wiki/Osgi`

[CVSHOME] cvshome.org: Distributions- und Dokumentationskanal des CVS-Konsortiums
URL [Juli 2004] `https://www.cvshome.org/`

[JMF-Guide] Sun Microsystems: „JMF Guide“, Dokumentation zu dem Java Media Framework
URL [August 2004] `http://java.sun.com/products/java-media/jmf/2.1.1/guide/`

[Java-GUI] Sun Microsystems: „Mixing heavy and light components“, Artikel über die Eigenschaften von Heavyweight- und Lightweight-Komponenten

URL [Juli 2004] `http://java.sun.com/products/jfc/tsc/articles/mixing/`

[XviD] xvid.org: Distributions- und Dokumentationskanal des XviD-Konsortiums
URL [August 2004] `http://www.xvid.org`

[FV-Spec] Heßling: „Handbuch zur Sensorschnittstelle Ortung (Sensorprotokoll)“, Dokumentation Version 1.19, März 2004, Robert Bosch GmbH intern

[NMEA] nmea.org: „National Marine Electronics Association“, Übergreifende Organistation des Global Positioning Systems
URL [August 2004] `http://www.nmea.org`

[NMEA-Spec1] gpsinformation.org: „NMEA data“, offene Spezifikation des NMEA-0183 Standards
URL [August 2004] `http://www.gpsinformation.org/dale/nmea.htm`

[NMEA-Spec2] Universität Regensburg: „Understanding NMEA 0183“, offene Spezifikation des NMEA-0183 Standards
URL [August 2004] `http://pcptpp030.psychologie.uni-regensburg.de/trafficresearch/NMEA0183/`

[Wiki-WGS84] de.wikipedia.org: „WGS84 Definition“, Wikipedia - Die freie Enzyklopädie
URL [August 2004] `http://de.wikipedia.org/wiki/WGS84`

[MS] Microsoft Corp.: „Microsoft SideWinder Force Feedback Wheel“, Lenkrad inklusuve Gas- und Bremspedal
URL [Juli 2004] `http://www.microsoft.com/hardware/sidewinder/FFB.asp`

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich zur Anfertigung der vorliegenden Arbeit keine anderen als die angegebenen Quellen und Hilfsmittel und keine nichtgenannte fremde Hilfe in Anspruch genommen habe.

Hildesheim, dem 28. Oktober 2004 Unterschrift: .....

A Spezifikationen

A.1 Datenformat NMEA[1]

NMEA steht für „National Marine Electronics Association“ und bezeichnet damit die nationale Marine Vereinigung der Vereinigten Staaten von Amerika. Sie ist eine übergeordnete Institution, die die Schirmherrschaft über den Aufbau des „Global Positioning System“ inne hat. Die Aufgaben der NMEA sind[2]:

- Unterstützen von Händlern von Marineelektronik in der Ausbildung, Kommunikation, im Training und in ihrer Zertifizierung.
- Stärken des industriellen Rufs auf dem Weltmarkt durch industrielle Standards.
- Bereitstellen eines guten Geschäftsmanagements und fairen Geschäftspraktiken für seine Mitglieder.
- Die Industrie ermutigen höher qualifiziertes technisches Personal einzustellen.
- Schult die Schifffahrtsöffentlichkeit durch Publikationen und Seminare um die sichere und ordnungsgemäße von elektronischer Marineausstattung zu gewährleisten.

Aus diesen Leitzielen ist das NMEA-0183-Protokoll entstanden, welches den Datenausgang eines GPS-Moduls zur Bestimmung von Ortungsdaten beschreibt. In diesem Protokoll können die GPS-Daten über eine serielle Schnittstelle (RS232), mit folgenden Spezifikationen im Textformat von einem GPS-Modul gelesen und geschrieben werden.

Typ	Wert
Baudrate	4800
Datenbits	8
Stopbits	1 oder 2
Parität	keine
Flusssteuerung	keine

Tabelle A.1: NMEA RS232 Konfiguration (Quelle: [NMEA-Spec2])

[1] Quelle: [NMEA].
[2] Quelle: [NMEA].

Ein handelsüblicher 'Personal Computer' kann somit an ein GPS-Ortungsmodul mit einer derartigen Konfiguration angeschlossen werden. Die standardisierte Frequenz für die Übertragung von einem Paar Ortungsdaten ist von der NMEA auf 1 Hz festegelegt. Einmal pro Sekunde werden von dem GPS-Modul Ortungsdaten an die RS232-Schnittstelle des Computers gesendet. Folgende Beispiele sind aus [NMEA-Spec1] entnommen und aufbereitet.

```
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
```

Die einzelnen Daten werden hierbei durch ein Komma separiert und können wie folgt interpretiert werden:

Eintrag	Bedeutung		
$GP	Typischer GPS-Präfix.		
GGA	Dieser String enthält zeitabhängige GPS Informationen.		
123519	UTC-Zeit zum Zeitpunkt der Ortsbestimmung.		
4807.038,N	Breitengrade: 48° 07.038' nördl. Breite		
01131.000,E	Längengrade: 11° 31.000' östl. Länge		
1	Qualitätsindex:	Wert	Bedeutung
		0	invalid
		1	GPS (SPS)
		2	DGPS
		3	PPS
		4	Real Time Kinematic
		5	Float RTK
		6	freigehalten (Kopplung) für 2.3
		7	manueller Eingabemodus
		8	Simulationsmodul
08	Anzahl von benutzten Satelliten zur Ortung.		
0.9	Horizontale Abweichung der Position.		
545.4,M	Höhe über Normal Null in Meter.		
46.9,M	Höhe über Erdgeoiden in Meter über WGS84 Ellipsoiden.		
leer	Zeit in Sekunden seit dem letzten DGPS Update.		
leer	DGPS Stations-ID.		
*47	Prüfsumme, beginnt immer mit *.		

Tabelle A.2: NMEA Spezifikationsbeispiel *$GPGGA*

Die meisten NMEA-Sätze enthalten den *$GPGGA*, welcher aktuelle zeitgebundene Ortungsdaten zur Verfügung stellt. Als weitere Informationen kann ein GPS-Modul auch folgende *$GPVTG* - Textzeile liefern:

```
$GPVTG,054.7,T,034.4,M,005.5,N,010.2,K
```

Eintrag	Bedeutung
$GP	Typischer GPS-Präfix.
VTG	Vector track and speed over ground. → Geschwindigkeitsinformationen.
054.7,T	Wahre Strecke, bereinigt.
034.4,M	Magnetische Strecke, bereinigt.
005.5,N	Geschwindigkeit über Grund in Knoten.
010.2,K	Geschwindigkeit über Grund in Km/h.

Tabelle A.3: NMEA Spezifikationsbeispiel *$GPVTG*

Außerdem beschreibt die offene Spezifikation NMEA-0183 noch weitere *$GP** Datensätze die bei einer Ortungsbestimmung durch ein GPS-Modul verwendet werden können.

A.2 Datenformat FV[3] [*vertraulich*]

Dieser Abschnitt ist nur dem internen Gebrauch in der Robert Bosch GmbH vorbehalten!

A.3 Dateiformat VSX

Das Dateiformat mit der Endung *.vsx (**V**ideo**S**im-**X**ML-Datei) ist ein proprietäres Dateiformat auf Basis des XML-Standards, welches im Rahmen dieser Diplomarbeit entwickelt wurde. Es speichert die gesamten Projektinformationen einer Fahrsimulation. Abbildung A.1 zeigt einen beispielhaft eine vsx-Datei, in der ein Projekt (Project), zwei Strecken (Track) mit je zwei Fahrten (Run) und deren spezifischen Definitionen enthalten sind.

[3] Quelle: [FV-Spec].

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- VideoSim Project file powered by Patrick Kosiol - property of Robert Bosch GmbH (FV/SLH)  -->
- <Project description="I am a videoSim project!">
  - <Track ID="0" description="Bosch --> Innenstadt" runCount="2">
      <Run ID="0" description="Tagfahrt 1"
        txtFile="C:\Programmierung\VideoSim\videoSim\storage\1ref\1084800400328.txt"
        aviFile="C:\Programmierung\VideoSim\videoSim\storage\1ref\1084800400328.avi"
        StartTime="13:08:33" EndTime="13:39:13" />
      <Run ID="1" description="Tagfahrt 2"
        txtFile="C:\Programmierung\VideoSim\videoSim\storage\1ref\1086262341906.txt"
        aviFile="C:\Programmierung\VideoSim\videoSim\storage\1ref\1086262341906.avi"
        StartTime="11:27:38" EndTime="11:41:15" />
    </Track>
  - <Track ID="1" description="Innenstadt --> Bosch" runCount="2">
      <Run ID="0" description="Tagfahrt 1"
        txtFile="C:\Programmierung\VideoSim\videoSim\storage\2ref\1084801457031.txt"
        aviFile="C:\Programmierung\VideoSim\videoSim\storage\2ref\1084801457031.avi"
        StartTime="13:39:13" EndTime="14:00:10" />
      <Run ID="1" description="Tagfahrt 2"
        txtFile="C:\Programmierung\VideoSim\videoSim\storage\2ref\1086263029500.txt"
        aviFile="C:\Programmierung\VideoSim\videoSim\storage\2ref\1086263029500.avi"
        StartTime="11:41:15" EndTime="11:53:59" />
    </Track>
  </Project>
```

Abbildung A.1: VSX Beispieldatei

Dieses Format ermöglicht die Strukturierung in Projekte, Strecken und Fahrten, wie schon in Abbildung 5.2 dargestellt. Zu Beginn einer VSX-Datei steht die Version und die verwendete Kodierung, danach kommt ein Kommentar, der darauf hinweist, dass eine Datei in diesem Format nur Eigentum der Robert Bosch GmbH sein kann, da dieses Format dort definiert ist. Der Folgende XML-Tag beschreibt das Projekt mit seinen Eigenschaften und kapselt in sich die enthaltenen Tracks (wiederum durch Track-Tags dargestellt). Jeder dieser Tracks besitzt auch spezifische Eigenschaften und kapselt in sich die vorhandenen Runs. Ein Run besteht dabei grundsätzlich von einer Datei mit den Videodaten (AVI-Datei) und einer Datei mit GPS-Daten (Text-Datei). Mit Hilfe einer VSX-Datei ist es möglich, komplette Strecken und Routen in dem VideoSim-Fahrsimulator zu erstellen, zu ändern, diese Änderungen dauerhaft zu speichern und wieder zu laden.

B Das Java Media Framework[1]

Das Java Media Framework ist eine Sammlung von Bibliotheken zur Verarbeitung multimedialer Daten. Diese reichen von einfachen Audio- und Videodateien über multimediale Datenströme bis hin zu diversen Codecs mit denen diese Daten komprimiert und dekomprimiert werden. Die PlugIn-Architektur [2] des JMF erlaubt es eine Software zur Verarbeitung von Multimediadaten an nahezu alle Bedürfnisse anzupassen.

Durch die Verwendung eines 'Panasonic DV Codecs', welcher frei als *pdvcodec.dll* mit zugehöriger Windowstreiber-Installationsdatei heruntergeladen werden kann, ist es möglich im JMF DigitalVideo(DV)-Daten zu verarbeiten. Diese Funktion unterstützt das Java Media Framework nicht von Hause aus. Mit Hilfe der Installation des pdv-Codecs unter Windows kann das JMF Digital-Video–Komprimier- und Dekomprimierfunktionen des Betriebssystems nutzen. Microsoft Windows 2000 stellt für diese Art der Videoverarbeitung die „Video for Windows"-Schnittstelle zur Verfügung. Dies ermöglicht die vorgestellte Lösung[3] zur synchronisierten Aufnahme von Video- und GPS-Daten.

[1] Quelle: [JMF-Guide].
[2] Vgl. Abschnitt 2.1.2 auf Seite 6.
[3] Vgl. Abschnitt 3.3 auf Seite 18.

B.1 JMF-Architektur und Datenmodelle

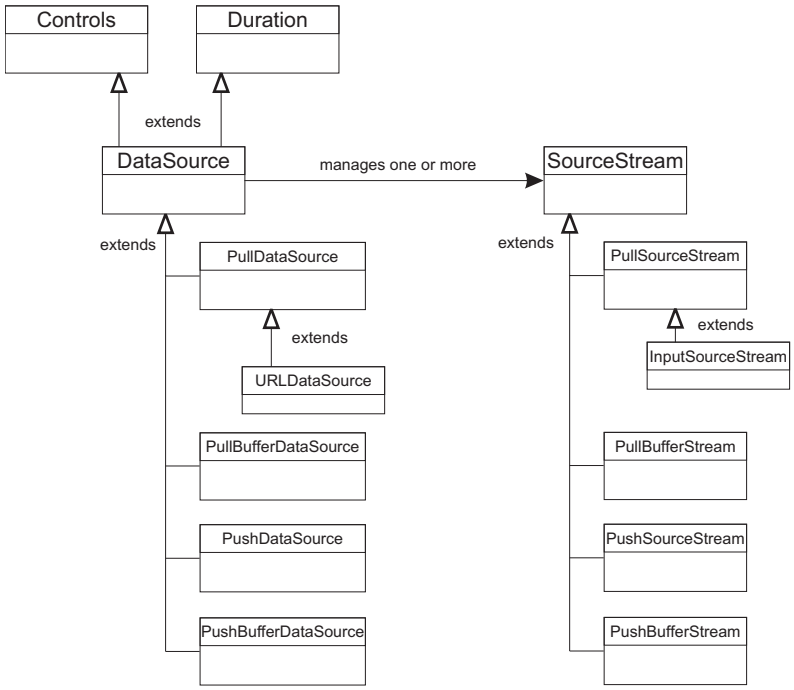


Abbildung B.1: JMF *DataSource* Architektur (Quelle: [JMF-Guide])

Abbildung B.1 zeigt eine *DataSource*, die von den beiden *Controls*- und *Duration*-Interfaces abgeleitet. Sie repräsentiert also eine Komponente, die die Dauer der enthaltenen Multimediadaten beinhaltet, sowie eine Komponente die ein Interface zur Kontrolle dieser Multimediadaten bereitstellt. Des Weiteren kann eine *DataSource* eine Vielzahl von *SourceStream* Objekten verwalten, welche die eigentlichen Multimediadaten darstellen. In der, im Rahmen dieser Diplomarbeit enstandenen Fahrsimulation besitzt jede *DataSource* genau einen *SourceStream*. Somit wird jeder erstellten *DataSource* genau eine Videofahrt zugeordnet. Aufbauend auf Abbildung 5.4[4] zeigt deren detailliertere Darstellung die Verwaltung dieser Datenquellen in dem UML-Klassendiagramm in Abbildung B.2.

[4] Vgl. Abschnitt 5.1 auf Seite 35.

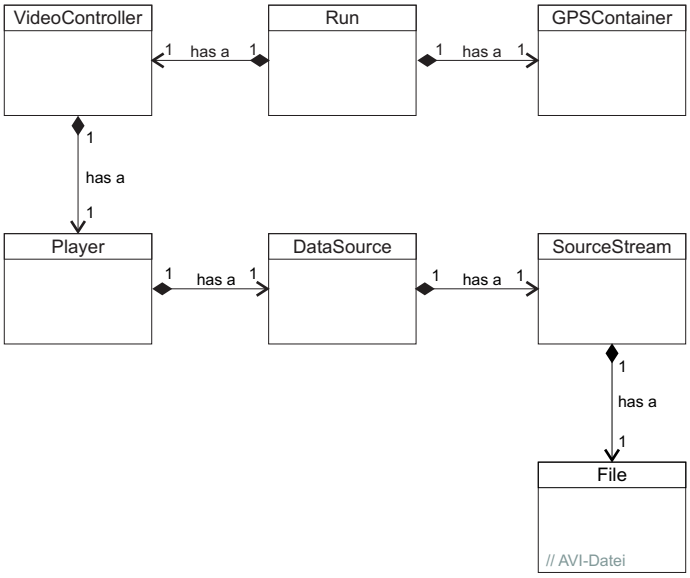


Abbildung B.2: UML-Klassendiagramm zur Verwaltung von JMF-Video-Daten in *VideoSim*

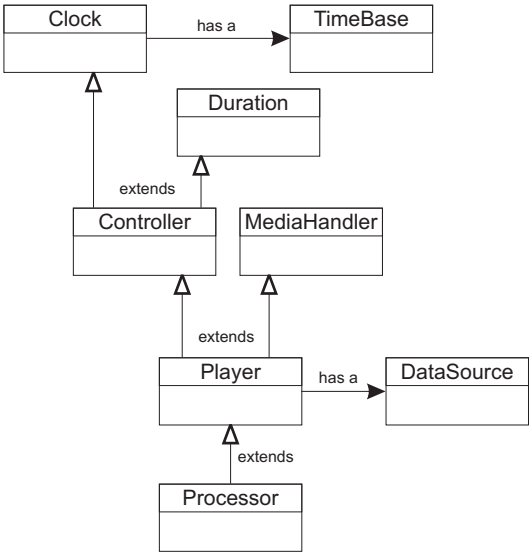


Abbildung B.3: JMF *Player/Processor* Architektur (Quelle: [JMF-Guide])

Abbildung B.3 zeigt die kontrollierende Multimediastruktur des JMF. Die wichtigsten Elemente hierbei sind die *Datasource*, der *Player* sowie der *Processor*. Ein *Player* besitzt immer eine *Datasource* als Quelle seiner Multimediadaten. Über dies kann statt eines *Players* ein *Processor* implementiert werden, welcher die Funktionalitäten des *Players*

erweitert[5]. Außerdem unterscheiden sich diese beiden Module in ihren Zustandsdiagrammen, wie die Abbildungen B.4 und B.5 zeigen.

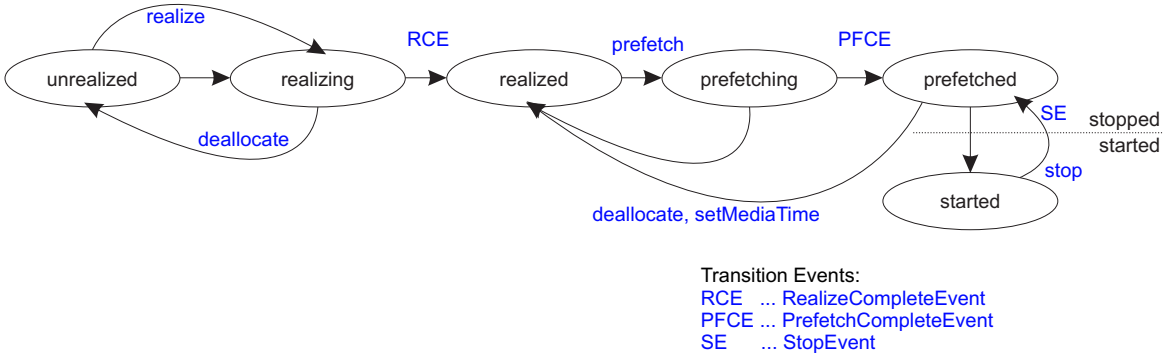


Abbildung B.4: Zustandsdiagramm *Player* (Quelle: [JMF-Guide])

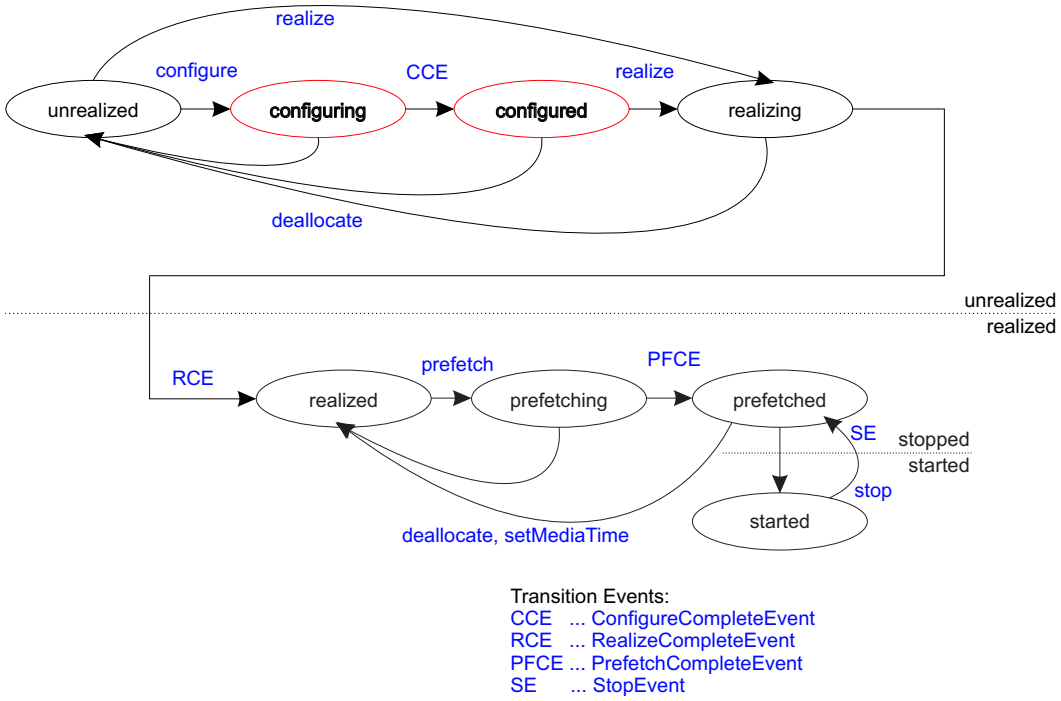


Abbildung B.5: Zustandsdiagramm *Processor* (Quelle: [JMF-Guide])

Die beiden Zustandsdiagramme von *Player* und *Processor* unterscheiden sich in dem Übergang von dem Status *unrealized* zu dem Status *realized*, also in dem Realisierungsprozess. Dabei muss ein *Processor* (aufgrund seiner erweiterten Funktionalität[6]) zuerst

[5] Vgl. Kapitel 4.2.1 auf Seite 23.
[6] Vgl. Kapitel 4.2.1 auf Seite 23.

konfiguriert werden, damit möglicherweise vorhandene Plug-Ins[7] registriert werden können. Die Verarbeitung von Multimediadaten ist in verschiedene Stadien eingeteilt. Ähnlich einer Kette können hier benutzerdefinierte Plug-Ins eingebunden werden, die dann bei der Verarbeitung eines Multimediastroms (in Form einer *DataSource*) berücksichtigt werden. Abbildung B.6 zeigt schematisch die verschiedenen PlugIn-Verarbeitungsstadien von multimedialen Daten innerhalb eines *Processors*.

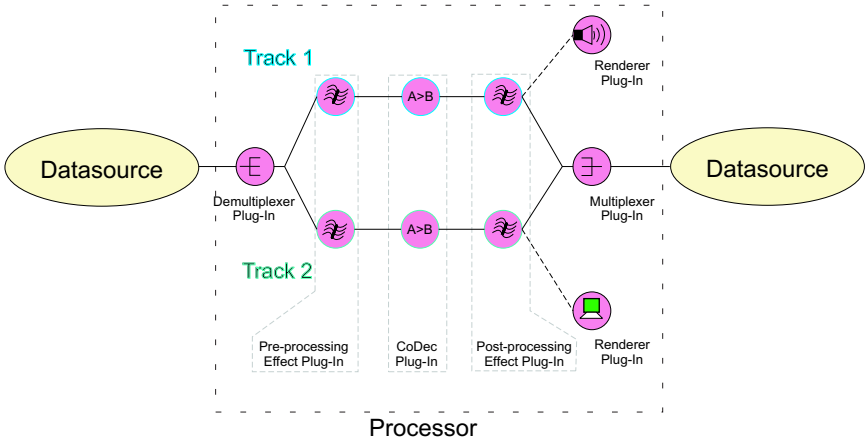


Abbildung B.6: Verarbeitung von Multimediadaten in einem *Processor* (Quelle: [JMF-Guide])

Tiefer gehende Informationenen über das Java Media Framework befinden sich im [JMF-Guide].

B.2 Darstellung von Videos in Java Components[8]

B.2.1 Heavyweight- und Lightweight-Komponenten

Das JMF ist in der Lage Videos als Multimediadaten zu verarbeiten und in einer GUI darzustellen. Bei der Darstellung von Videobildern über einen Player bzw. einen Processor besteht die Möglich zu entscheiden, ob diese in einer *Heavyweight*- oder in einer *Lightweight*-Komponente dargestellt werden sollen. Dabei treten erhebliche Unterschiede zwischen diesen beiden Darstellungsmöglichkeiten in Bezug auf Parameter, wie Geschwindigkeit und Prozessorauslastung oder Overlayeffekten auf. Dabei werden die Standard-Java-Implementierungen von AWT-Komponenten als *Heavyweight* und die von Swing-Komponenten als *Lightweight* bezeichnet. Swing-Komponenten basieren in diesem Zusammenhang auf der AWT-Architektur, welches eines der Hauptziele bei der Entwicklung war. Eine bedeutende Möglichkeit dabei sollte sein, bestehende AWT-Komponenten-basierte

[7] Vgl. Abbildung 2.2 auf Seite 7.
[8] Quelle: [Java-GUI].

Programme leicht nach Swing portieren zu können. Da Swing und AWT die gleiche Infrastruktur besitzen, ist es also auch möglich AWT- und Swing-Komponenten in einem Programm zu vermischen.

Ein Hauptunterschied zwischen AWT- und Swing-Komponenten besteht darin, dass jede AWT-Komponente seine eigene native Darstellungsressource (auch „peer“ genannt) besitzt. Eine Swing-Komponente im Gegensatz, ist eine Komponente, die sich die Darstellungsressource seines „Vorfahren“ (also dessen Komponente, die diese Komponente erzeugt hat) „borgt“. Dies bedeutet, dass eine Swing-Komponente keine native Darstellungsressource für sich allein hat und sie somit „lighter“ ist. Es werden Ressourcen geteilt, die bei AWT-Komponenten für jede einzelne angelegt werden.

Vorteile bei dem Gebrauch von Swing-Komponenten:

- Effizientere Nutzung von Ressourcen
- Höhere Konsistenz bei Plattformunabhängigkeit (da Swing in allen Javaimplementierungen integriert wurde)
- Saubere look-and-feel Integration (mit Swing ist es möglich Gruppen von Komponenten ein bestimmtes look-and-feel zuzuordnen)

AWT-Komponenten sind *heavyweight* und Swing-Komponenten sind *lightweight* (mit der Ausnahme der Top-Level-Komponenten JWindow, JFrame, JDialog und JApplet). Es treten noch weitere Unterschiede zwischen diesen beiden Gruppen von GUI-Elementen auf, die bei deren Einsatz beachtet werden müssen:

- Lightweight-Komponenten können transparente Pixel besitzt; Heavyweight-Komponenten können nicht transparent sein.
- Lightweight-Komponenten müssen nicht unbedingt rechteckig sein, aufgrund der Möglichkeit von transparenten Pixeln; Heavyweight-Komponenten sind immer rechteckig.
- MouseEvents bei Lightweight-Komponenten werden bis zu dessen „Vorfahren“ durchgereicht; Bei Heavyweight-Komponenten geschieht dies meist nicht.
- Wenn eine Lightweight-Komponente eine Heavyweight-Komponente überlappt, dann ist die Heavyweight-Komponente immer oben und die Lightweight-Komponente wird überdeckt.

B.2.2 AWT oder Swing für die Darstellung der Videos

Da das JMF die Darstellung des Videos in Swing- oder AWT-Komponenten zur Verfügung, ist eine Abwägung dieser beiden Möglichkeiten nötig. Die Darstellung des Videos

in einer Swing-Komponente hätte den Vorteil, andere transparente Swing-Komponenten über den Darstellungsbereich zu legen, um dem Fahrer anderweitige Informationen zukommen zu lassen. Ein Problem der Darstellung in Swing-Komponenten ist jedoch die Lightweight-Eigenschaft. Also die Tatsache, dass dabei Ressourcen geteilt werden. Ein Video ist eine aufwendige Darstellung von ca. 25 Bildern pro Sekunde (im Idealfall), bei der es nicht von Vorteil ist, wenn die Ressourcen des Darstellungsbereichs mit noch anderen GUI-Elementen geteilt werden. Aufgrund von Testläufen während der Evaluation der Fahrsimulation, hat sich herausgestellt, dass Swing-Komponenten in etwa das doppelte an CPU-Last benötigen als AWT-Komponenten, um das selbe Video darzustellen. Dieser signifikante Unterschied lässt nur einen Schluss in der Wahl zwischen AWT und Swing zu: Auch wenn Swing bei der Flexibilität im Design der Darstellung einen Vorteil hat, AWT ist die eindeutig bessere Lösung für das Problem der Darstellung von Videos in Java-GUI-Komponenten. Da der Gewinn bei der Reduzierung der CPU ein erhebliches Kriterium für die *Entwicklung einer videobasierten Fahrsimulation auf Basis realer Video- und GPS-Daten* ist.

C Steuerung von Multimedia Streams unter Java

C.1 Anbinden von DirectInput-Geräten über das Native Interface von Java

Eine Java/Windows-Architektur stellt in der Standardimplementierung keine Bibliotheken zur Anbindung von DirectInput-Geräten zur Verfügung. DirectInput ist eine Schnittstelle für Eingabegeräte innerhalb des DirectX-Packets von Microsoft. DirectX stellt verschiedenste Funktionen und Schnittstellen (API) den Programmierern zur Verfügung wie:

- DirectInput: Eingabegeräte wie beispielsweise Joysticks, Gamepads, Lenkräder und Tastaturen
- DirectShow: Ausgabeschnittstellen wie DirectDraw oder Direct3D
- DirectSound: Audioausgabegeräte wie beispielsweise Soundkarten zur Ausgabe von Wavedateien
- DirectMusic: Audioausgabegeräte wie beispielsweise Soundkarten mit Synthesizerfunktionen, MIDI-Mapper oder auch Software-Synthesizer
- DirectPlay: Kommunikationsgeräte zur Verbindung einer Applikation mit einer anderen zum Datenaustausch, beispielsweise über die Netzwerke auf Basis von TCP/IP

Abbildung C.1 zeigt die konzeptionelle Anbindung solcher DirectInput-Geräte über die Schnittstelle von Windows. Java benötigt eine Wrapper-Bibliothek, um auf DirectInput-Geräte wie Lenkräder und Gamepads zugreifen zu können.

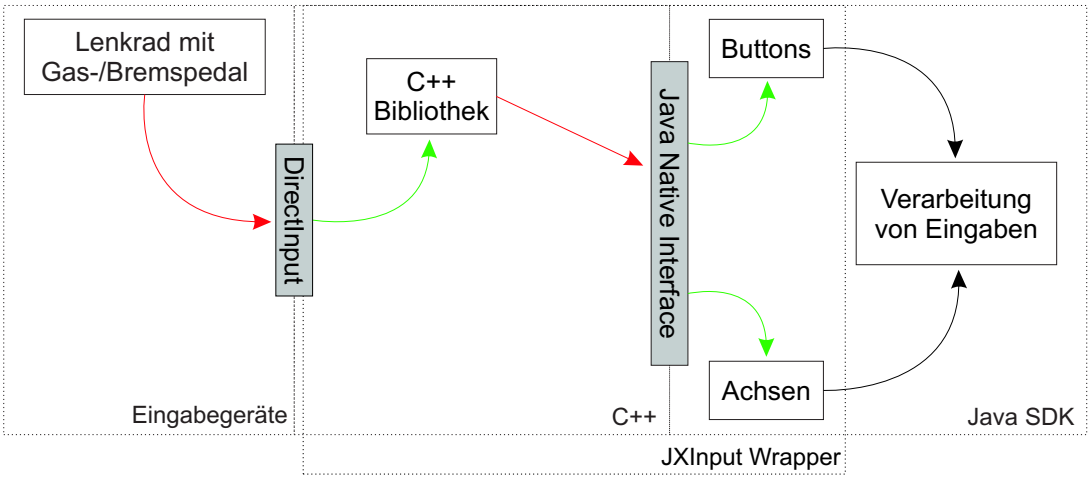


Abbildung C.1: JNI-DirectInput-Achitektur

Der *JXInput Wrapper* stellt einen Großteil der Funktionalitäten der DirectInput-Schnittstelle über das Java Native Interface einem Java Programm zur Verfügung. Mit einer kontinuierlichen Abfrage der Daten des DirectInput-Gerätes wird jede Veränderung (z.B. Gaspedal betätigen oder Button drücken) registriert. Daraus werden verschiedene Aktionen abgeleitet, wie beispielsweise das Beschleunigen des Videos bei dem Betätigen des Gaspedals.

Abbildung C.2 zeigt das 'Microsoft SideWinder ForceFeedback Wheel', welches als DirectInput-Gerät zur Steuerung der Fahrsimulation verwendet werden kann.

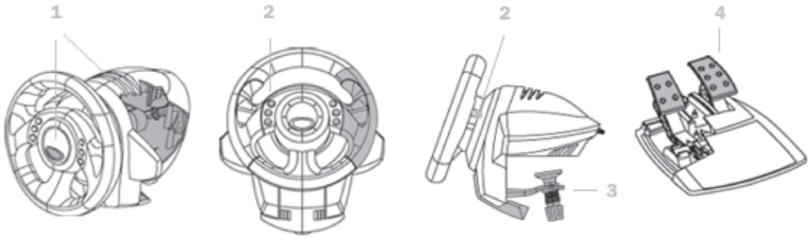


Abbildung C.2: Microsoft SideWinder ForceFeedback Wheel (Quelle: [MS])

Nr.	Bedeutung
1	ForceFeedbackschalter und Motor
2	8 frei programmierbare Funktionstasten
3	Tischbefestigung
4	Gas- und Bremspedal

Tabelle C.1: Legende zu Abbildung C.2

C.2 Keyboardsteuerung der Fahrsimulation

Die dynamische Steuerung der Fahrsimulation kann nicht nur über das DirectX-Interface DirectInput erreicht werden. Durch das Abgreifen bestimmter Tastendrücke auf der Tastatur kann dies auch geschehen, dabei dienen die Cursor-Tasten (hoch/runter) zur Geschwindigkeitsregelung. Außerdem kann ein weiteres Lenkrad an die Fahrsimulation angeschlossen werden. Dessen Multifunktionstasten nicht über das DirectInput-Interface von Microsoft Daten senden, sondern dies über definierte Tastatursignale bewerkstelligen. Demnach können die auf Abbildung C.3 dargestellten Fuktionstasten an der Lenkradinnenfläche als Tasten einer Tastatur angesehen werden.

Abbildung C.3: BMW-Multifunktionslenkrad der Baureihe E38

Die mittig angebrachte Hupe beispielsweise, wird wie das Drücken der Taste F7 behandelt. Alle anderen Tasten an diesem Lenkrad sind ebenfalls auf die restlichen F-Tasten einer Standard-Computer-Tastatur abgebildet.

C.3 Proportionalitäten im Bereich der CPU-Auslastung

Wie im Abschnitt 4.3 auf Seite 29*ff.* dargestellt, besteht ein Zusammenhang zwischen der CPU-Auslastung des Computers und der eingestellten Abspielgeschwindigkeit eines Videos im Java Media Framework. Aufgrund der Eigenschaften des Videos versucht die Software bei einer höheren Abspielrate auch mehr Videobilder pro Sekunde darzustellen. Dies erhöht den Berechnungsaufwand und kann den Prozessor eines Rechners überlasten. Abbildung C.4 zeigt diesen Zusammenhang zwischen der CPU-Auslastung und der Abspielgeschwindigkeit.

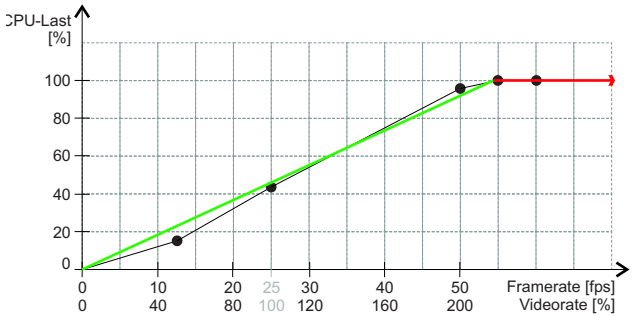


Abbildung C.4: Proportionalitäten im Bereich der CPU-Auslastung

Aus dieser Graphik lässt sich eine direkte Proportionalität zwischen den dargestellten Achsen ableiten.
Unter der Bedingung $Videorate < ca.\ 210\%$ gilt[1]:

$$CPULast \sim Framerate$$

Analog hierzu gilt auch folgende Proportionalität:

$$CPULast \sim Videorate$$

Unter der Bedingung $Videorate > ca.\ 210\%$ gilt[2]:

$$CPULast \equiv MAXIMUM \equiv 100\%$$

Da die CPU-Auslastung nicht über die 100%-Marke steigen kann, verändert sich dieser Wert ab einer Videoabspielrate von 210% und mehr. Dies würde einen Versuch der Darstellung mehr als 52 Bilder pro Sekunde durch das Java Media Framework bedeuten. Diese Daten sind Erfahrungswerte, die mit Hilfe eines Computersystems mit folgender Konfiguration erfasst wurden: PC IBM NetVista, Pentium 4 mit 2.0 Ghz, 512 MB Ram, 20 GB Festplatte, Intel OnBoard-Graphikkarte.

[1] In Abbildung C.4 grün hervorgehoben.
[2] In Abbildung C.4 rot hervorgehoben.

D Daten

D.1 Aufgenommene Strecken und Routen

Zum Zeitpunkt der Fertigstellung dieser Diplomarbeit wurde eine Vielzahl von Strecken mit unterschiedlichen Fahrten aufgenommen. Abbildung D.1 zeigt schematisch die logische Darstellung dieser Strecken.

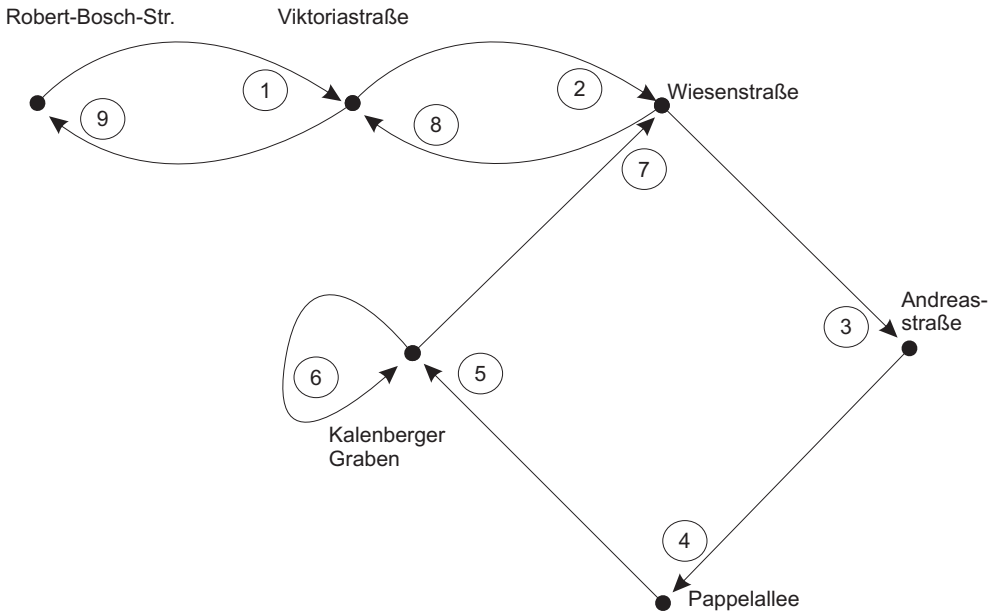


Abbildung D.1: Logische Streckenübersicht

Die dargestellten Punkte zu denen die Pfeile zeigen, repräsentieren Schnittpunkte an denen sich Strecken treffen, also an denen Strecken beginnen oder enden. Die Bezeichnungen an diesen Punkten sind die Straßennamen dieser geographischen Stellen in Hildesheim. Die Pfeile zwischen diesen Punkten sind die einzelnen Strecken. Über jede dieser Strecken kann eine unterschiedliche Anzahl von Fahrten vorliegen[1]. An den Pfeilen sind die Strecken durchnummeriert anhand derer eine Auflistung der Strecken wie in Tabelle D.1 erfolgen kann.

[1] Vgl. mit Abschnitt 5.1 auf Seite 35.

Nr.	Startpunkt	Endpunkt
1	Robert-Bosch-Str.	Viktoriastraße
2	Viktoriastraße	Wiesenstraße
3	Wiesenstraße	Andreasstraße
4	Andreasstraße	Pappelallee
5	Pappelallee	Kalenberger Graben
6	Kalenberger Graben	Kalenberger Graben
7	Kalenberger Graben	Wiesenstraße
8	Wiesenstraße	Viktoriastraße
9	Viktoriastraße	Robert-Bosch-Str.

Tabelle D.1: Streckenauflistung aus Abbildung D.1 und D.2

Abbildung D.2: Graphische Streckenübersicht

Eine detaillierte graphische Darstellung aller Strecken wird in den folgenden neun Abbildungen gezeigt:

Abbildung D.3: Graphische Streckenübersicht - Strecke 1

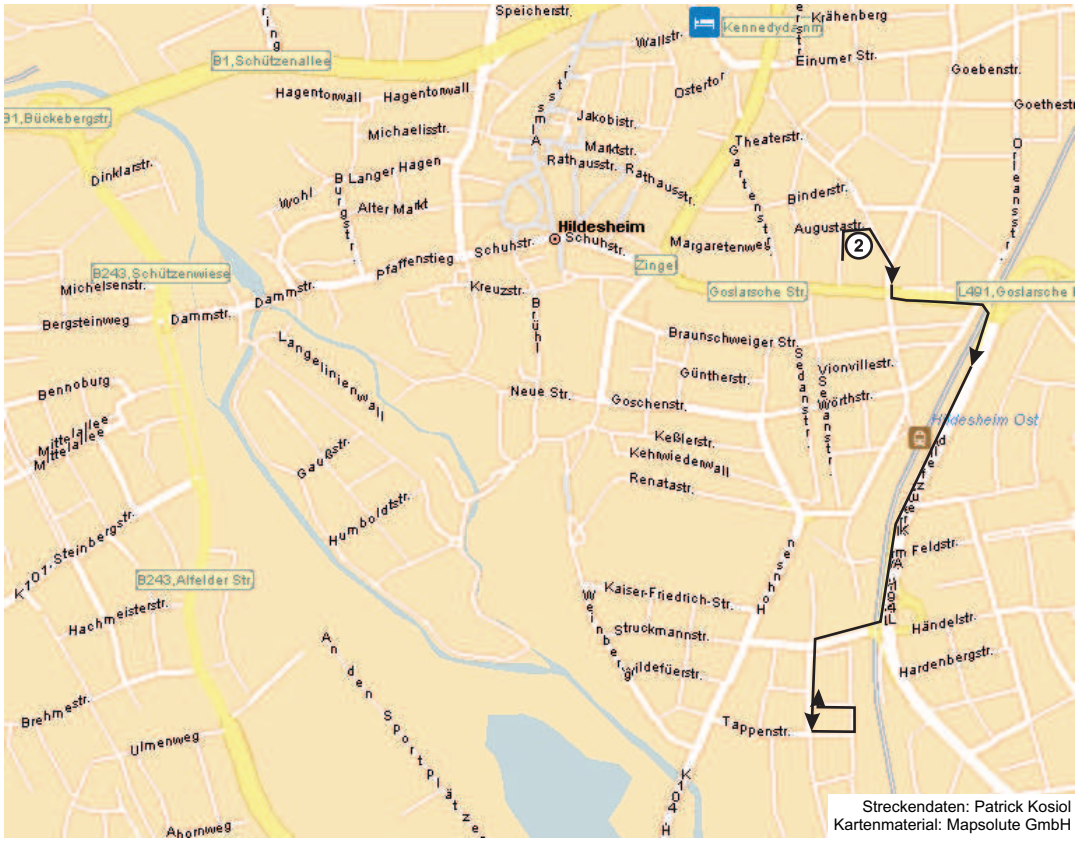


Abbildung D.4: Graphische Streckenübersicht - Strecke 2

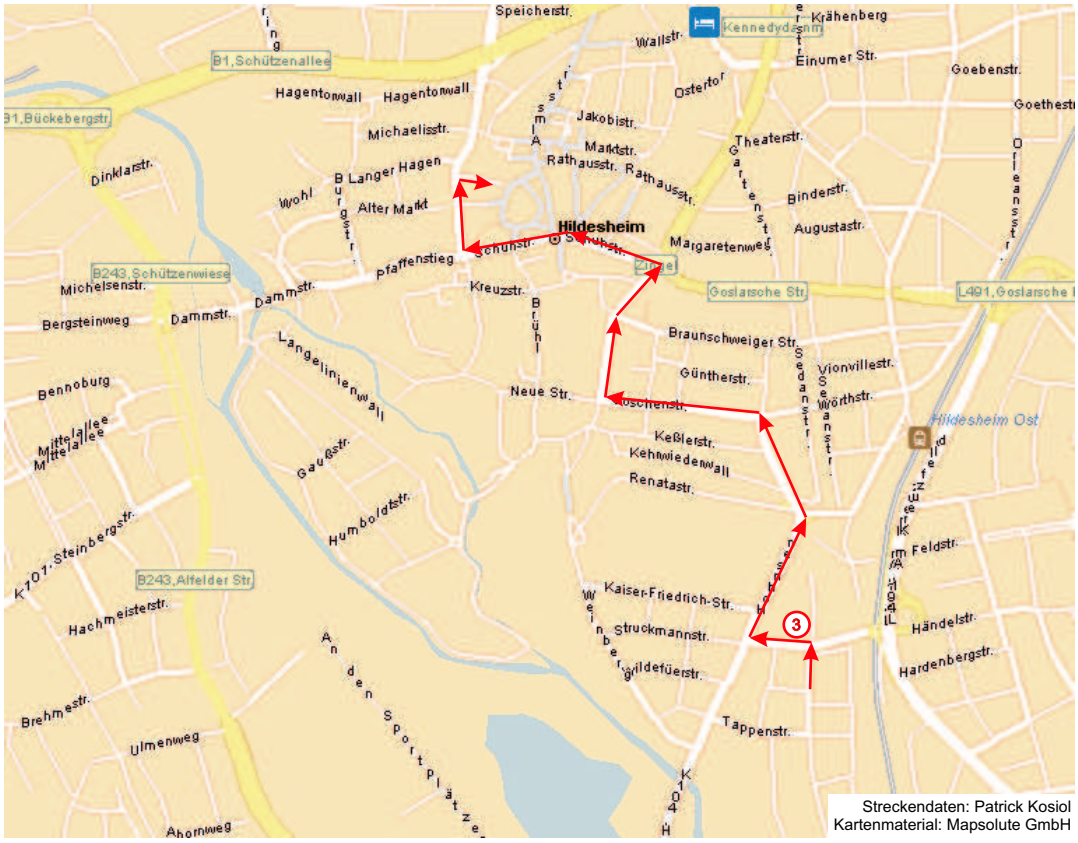


Abbildung D.5: Graphische Streckenübersicht - Strecke 3

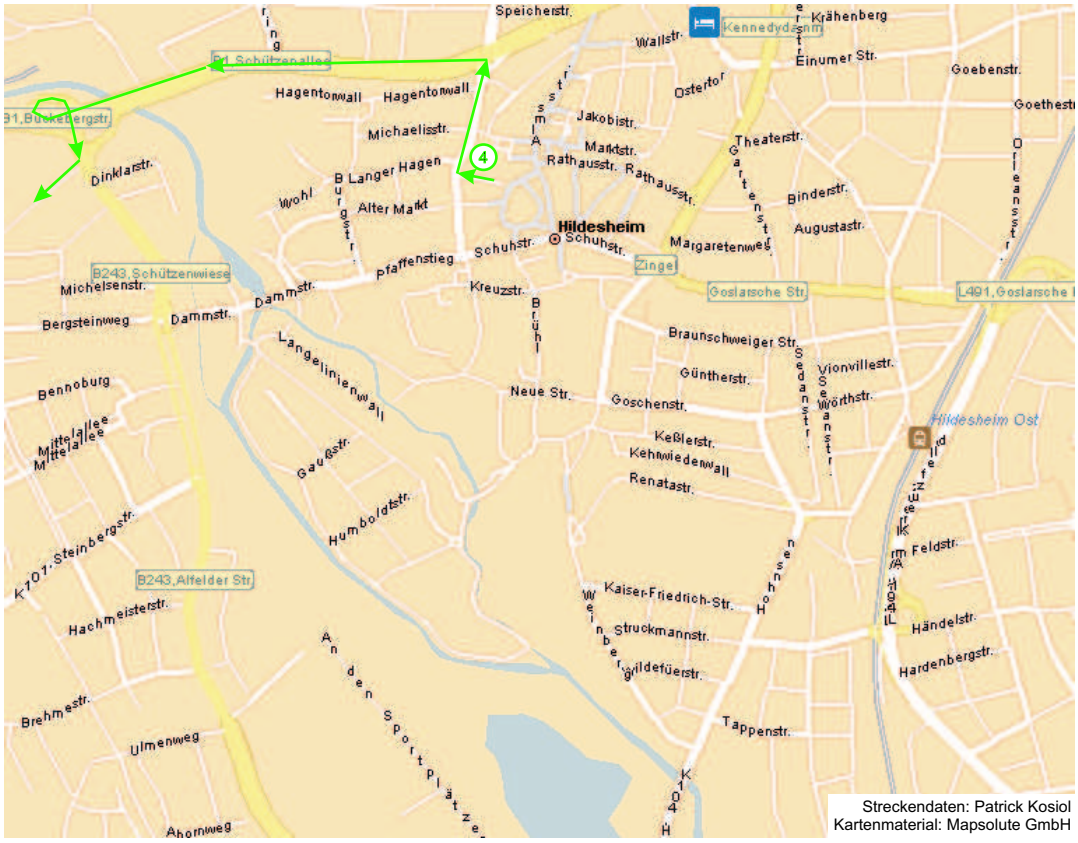


Abbildung D.6: Graphische Streckenübersicht - Strecke 4

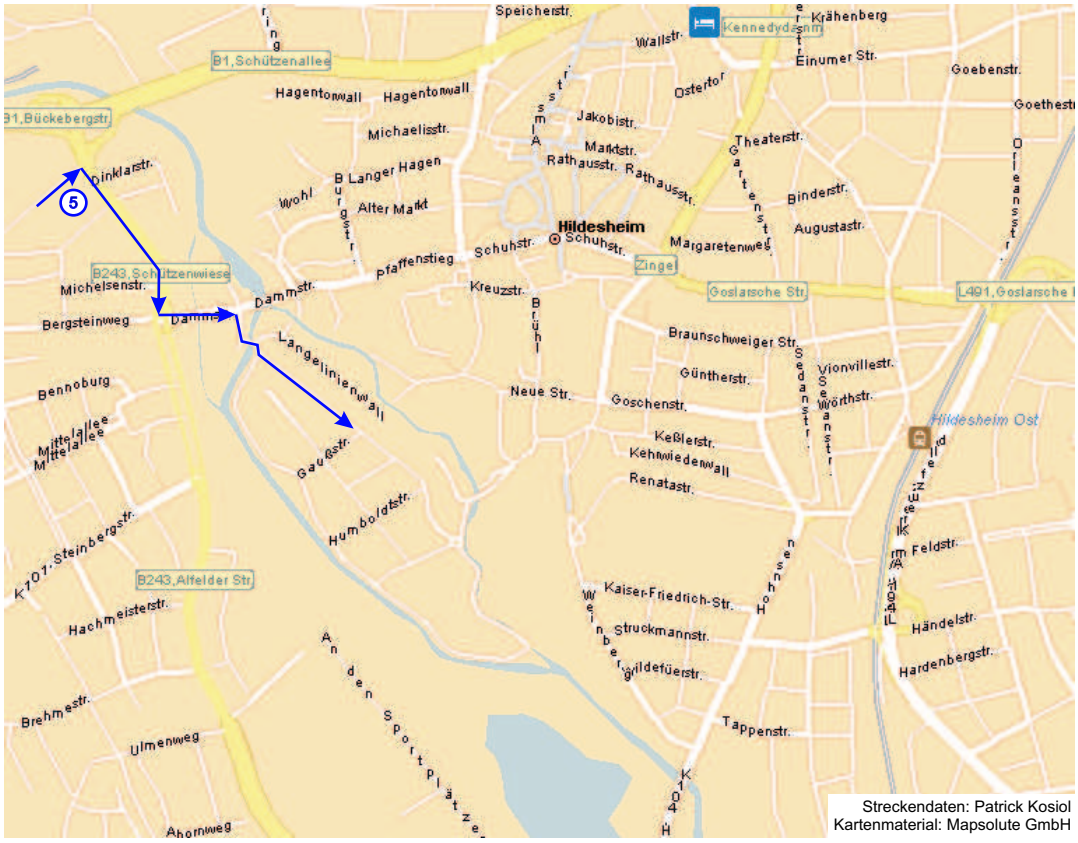


Abbildung D.7: Graphische Streckenübersicht - Strecke 5

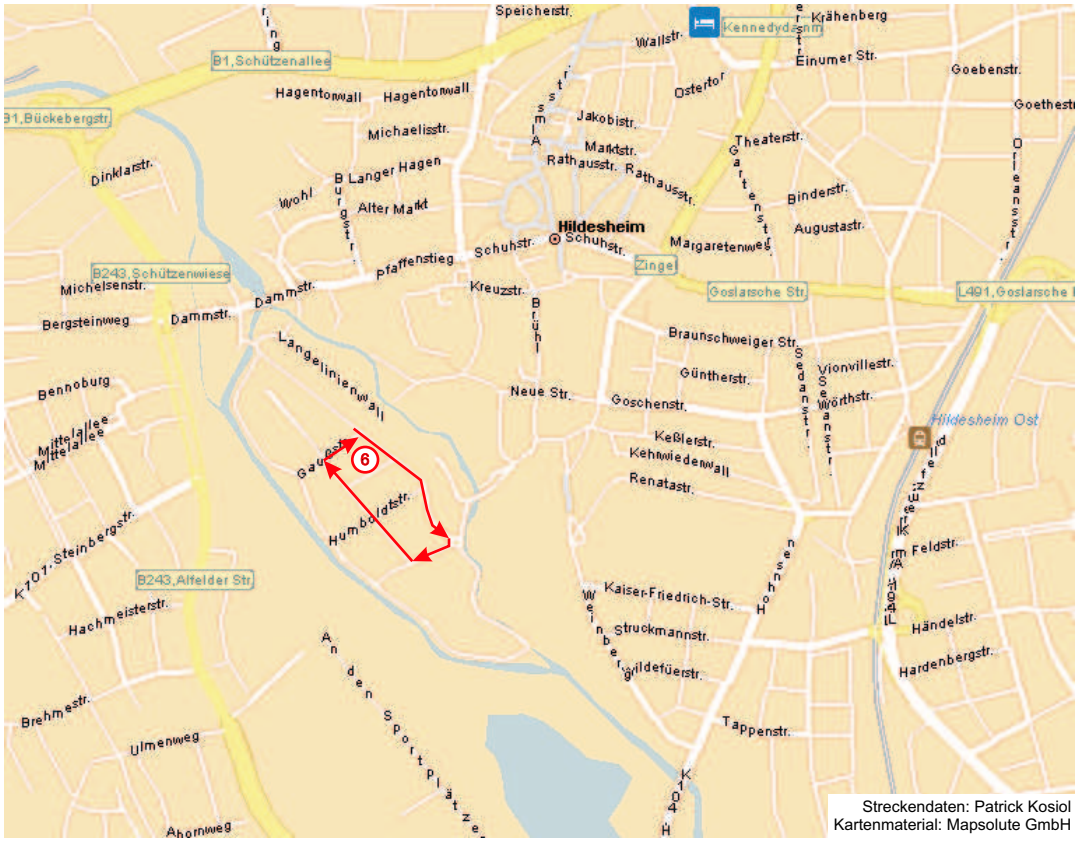


Abbildung D.8: Graphische Streckenübersicht - Strecke 6

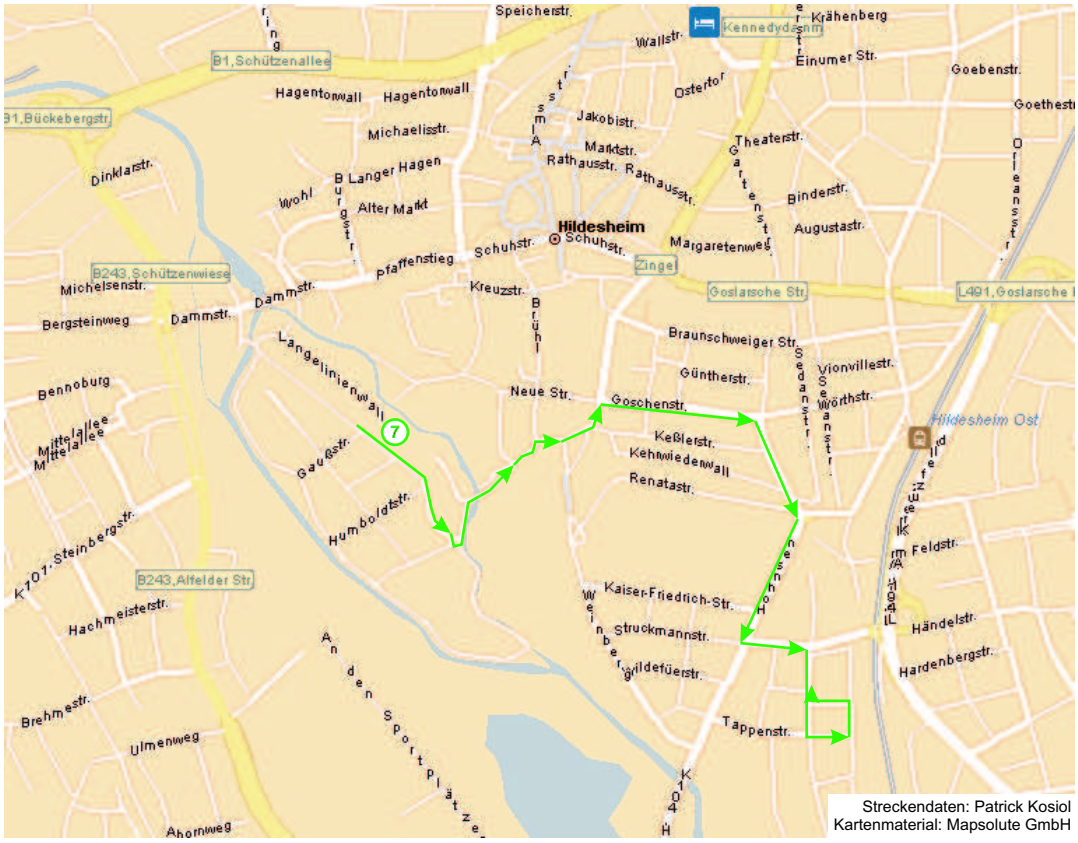


Abbildung D.9: Graphische Streckenübersicht - Strecke 7

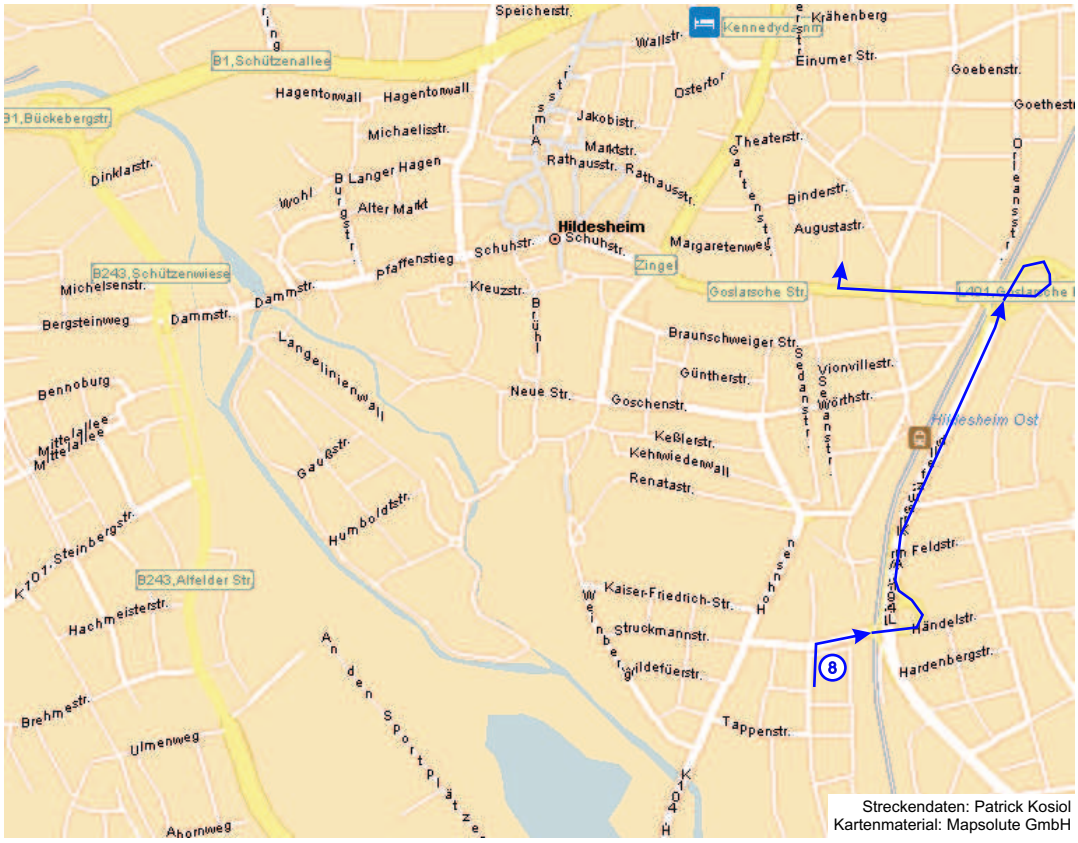


Abbildung D.10: Graphische Streckenübersicht - Strecke 8

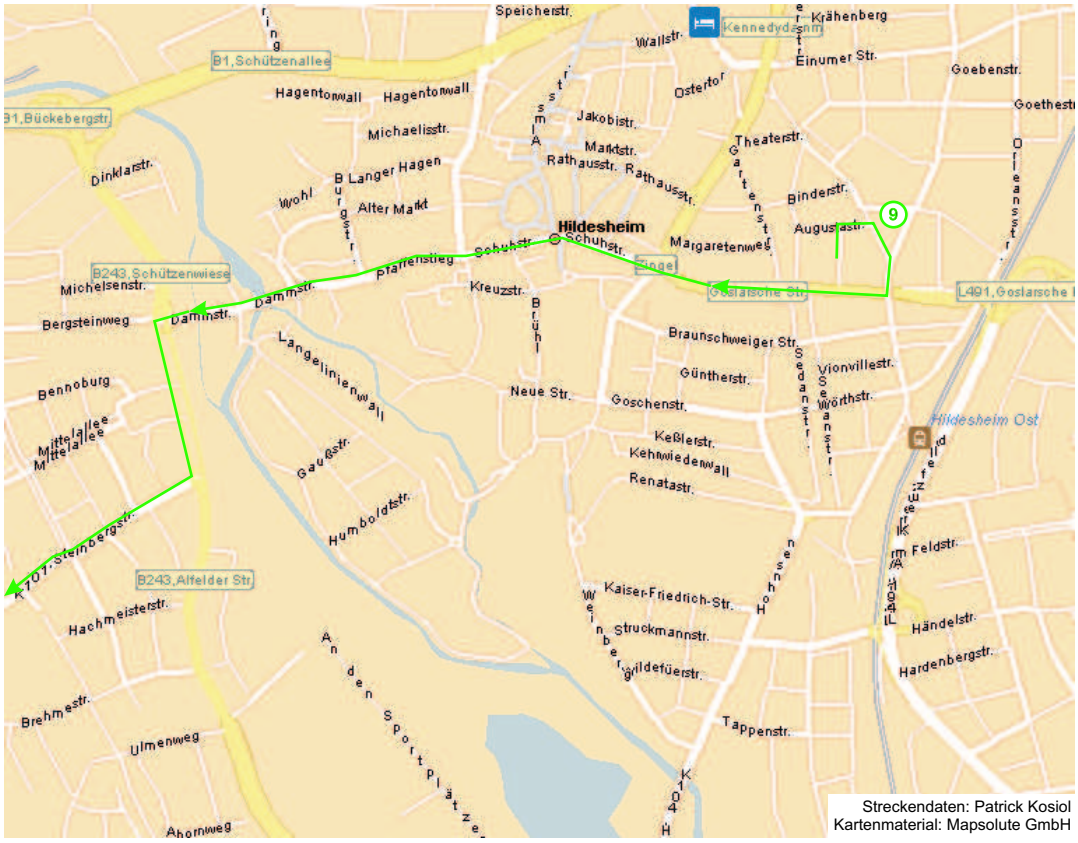


Abbildung D.11: Graphische Streckenübersicht - Strecke 9