

Prüfer: Prof. Dr. K. Rothermel  
Betreuer : Dipl.-Inf. Alexander Leonhardi  
Beginn am: 1. Februar 2001  
Beendet am: 31. Juli 2001  
CR-Klassifikation: C.2.1, C.4, D.4.4, D.4.8, G.1.2

Diplomarbeit-Nr. 1919

**Untersuchung von  
Koppelnavigations-Protokollen  
für die Übertragung von  
Positionsinformationen**

**Christian Nicu**

## **Kurzzusammenfassung**

Bei ortsbezogenen Diensten ist es für den Lokationsserver wichtig, die Positionsinformationen der Mobilien Objekte mit einer hohen Genauigkeit zu kennen. Mit den herkömmlichen Verfahren werden dazu sehr viele Nachrichten an den Server benötigt. Durch den Einsatz von Koppelnavigation ist es bei gleicher Genauigkeit möglich, die Kommunikation zur Positionsmitteilung zwischen Client und Server deutlich zu reduzieren.

Im Rahmen der Arbeit werden verschiedene Koppelnavigationsverfahren vorgestellt und bewertet. In diesem Zusammenhang werden wichtige Kernpunkte besprochen, die bei einer konkreten Umsetzung zu beachten sind, und Lösungen für in der Praxis auftretende Probleme aufgezeigt.

Schließlich werden ein einfaches und ein komplexes Verfahren (Lineares Fortsetzen und Kartenbasierte Koppelnavigation) genauer betrachtet. Beide wurden implementiert und ihre Effizienz anhand von Simulation einem Verfahren gegenübergestellt, welches keine Koppelnavigation benutzt. Dabei wurden verschiedene Bewegungsmuster unterschieden: Autobahn-, Überland-, Stadtfahrten und Fußgänger.

Die Ergebnisse bestätigen, daß durch den Einsatz von Koppelnavigation je nach Verfahren und Bewegungsmuster bis zu 91% der Kommunikation eingespart werden kann!

# Inhaltsverzeichnis

1	Einleitung.....	4
2	Grundlagen.....	6
2.1	Koppelnavigationsprotokolle.....	6
2.2	Taxonomie der Verfahren.....	8
2.2.1	Fortsetzungsverfahren.....	9
2.2.1.1	Lineare Fortsetzung.....	9
2.2.1.2	Krümmungsapproximation.....	10
2.2.2	Kartenbasierte Verfahren.....	10
2.2.2.1	Einfaches kartenbasiertes Verfahren.....	10
2.2.2.2	Routenplanung.....	11
2.2.2.3	Kartenbasiert mit Wahrscheinlichkeiten.....	11
2.2.3	Historienbasierte Koppelnavigation.....	11
2.3	Relevante Koordinatensysteme.....	12
3	Diskussion der Protokolle.....	14
3.1	Bewertung der Verfahren.....	14
3.1.1	Fortsetzungsverfahren.....	15
3.1.2	Historienbasierte Koppelnavigation.....	16
3.1.3	Kartenbasierte Verfahren.....	17
3.1.4	Bewertung und Zusammenfassung.....	20
3.2	Diskussion der praktischen Umsetzung.....	21
3.2.1	Einfluß von Rechenungenauigkeiten.....	21
3.2.2	Grad der Übereinstimmung der Daten zwischen Client und Server.....	23
3.2.3	Übertragung der Daten zum Server.....	25
3.2.4	Erkennung von Funkschatten oder Verbindungsabbrüchen.....	27
3.2.5	Grad der Uhrensynchronität.....	29
3.3	Ausblick.....	31
4	Gemeinsame Grundbestandteile aller Verfahren.....	32
4.1	Struktur von Updatenachrichten.....	32
4.2	Abstrakte Klasse DRAlgorithm.....	33
4.3	Bestimmung der neuen Soll-Position.....	35
4.4	Mobiles Objekt (Client).....	36
4.5	Lokationsdienst (Server).....	36
4.5.1	Datenbestand.....	37
4.5.2	Beantwortung von Lokationsanfragen.....	38
4.5.3	Verarbeitung von Updatenachrichten.....	38
4.5.4	Weitere Aufgaben des Servers.....	39
4.6	Vorschlag für ein Protokoll.....	39
4.6.1	Bestandteile.....	39
4.6.2	Verbindungsaufbau.....	40
4.6.3	Während eine Verbindung besteht.....	43

---

4.6.4 Neuaushandlung der Parameter.....	43
4.6.5 Ende der Verbindung.....	44
<b>5 Lineares Fortsetzen.....</b>	<b>45</b>
5.1 Linienglättung.....	45
5.1.1 Mathematische Grundlagen.....	46
5.1.1.1 Gaußverfahren.....	46
5.1.1.2 Geraden in Vektordarstellung.....	47
5.1.1.3 Verallgemeinerung des Verfahrens.....	48
5.1.2 Iterationslösung.....	49
5.1.3 Optimierung des Änderungsverhaltes des Richtungsvektors.....	50
5.2 Ablauf des Verfahrens.....	52
<b>6 Kartenbasierte Koppelnavigation.....</b>	<b>55</b>
6.1 Datenstruktur für elektronische Karten.....	55
6.1.1 Bestandteile.....	55
6.1.1.1 Knoten.....	55
6.1.1.2 Kanten.....	57
6.1.2 Formalisierung.....	58
6.1.3 Bestehende Standards.....	59
6.2 Map Matching.....	60
6.2.1 Einführung in die Problematik.....	60
6.2.1.1 Unkritischer Fall.....	60
6.2.1.2 Kritische Fälle.....	61
6.2.2 Geometrie einer Kante.....	63
6.2.3 Startposition.....	64
6.2.4 Kantenübergang.....	67
6.2.4.1 Erkennung von falschem Matching.....	68
6.2.4.2 Finden einer neuen Kante.....	68
6.2.5 Verlassen des Kartenbereichs.....	71
6.2.6 Weitere Map Matching Verfahren.....	73
6.3 Ablauf des Verfahrens.....	74
<b>7 Simulation.....</b>	<b>80</b>
7.1 Datenerfassung.....	80
7.2 Kartendaten.....	81
7.3 Simulationsumgebung.....	81
7.3.1 Strukturierung des Simulators.....	82
7.3.2 Struktur der Kartendaten.....	84
7.3.3 Graphische Datenpräsentation.....	85
7.3.4 Kenndaten der Simulation.....	86
7.4 Ergebnisse.....	87
7.4.1 Lineares Fortsetzen.....	88
7.4.2 Kartenbasierte Koppelnavigation.....	90
7.4.3 Bewertung.....	92

---

7.5 Fehlerabschätzung.....	93
7.5.1 Einordnung anderer Verfahren.....	93
7.5.2 Mögliche Probleme.....	94
7.5.2.1 Meßsystem.....	94
7.5.2.2 Kommunikation.....	94
7.5.2.3 Vorgegebene Genauigkeit.....	95
7.5.3 Verbesserungen der Verfahren.....	95
7.5.3.1 Berücksichtigung der Beschleunigung.....	95
7.5.3.2 Map Matching.....	97
8 Fazit und Zusammenfassung.....	99
9 Begriffslexikon.....	101
10 Quellenverzeichnis.....	104
Anhang.....	106
Einzelne Ergebnisse .....	106
Gesamtresultate.....	118
Diagramme.....	121

# 1 Einleitung

---

Ortsbezogene Dienste (Location Based Services) gelten als vielversprechendes Anwendungsgebiet für Mobilkommunikationssysteme der 3. Generation. Auf der CeBit 2001 hat beispielsweise der Mobilfunkprovider Viag-Interkom einen Dienst namens „M-Kompaß“ vorgestellt [VIAG], bei dem abhängig vom Aufenthaltsort das Handy aktuelle Informationen aus der Umgebung (Hotels, Restaurants, Kino) übermittelt bekommt. Zwar benutzt dieser Service zur Ortsbestimmung des Handys lediglich die Infrastruktur des GSM-Netzes, das eine Ortung nur sehr ungenau ermöglicht, aber es zeigt, daß sich bereits erste Anwendungen ortsbezogener Dienste entwickeln. Andere Netzbetreiber haben inzwischen nachgezogen und ebenfalls ortsbezogene Dienste angeboten: [TMOT], [D2MAN].

Außer diesen Grunddiensten sind weitere Anwendungen für Location Based Services vorstellbar. Man kann Systeme entwickeln, die den Aufenthaltsort von einer Vielzahl von Mobilobjekten betrachten. So wäre es für eine Einsatzzentrale der Polizei von Vorteil stets zu wissen, wo sich die Streifenwagen befinden. Im Falle eines Einsatzes könnte automatisch der nächstgelegene Wagen ausgesucht werden und zum Einsatzort geschickt werden. Zusätzlich könnte man noch eine Routenplanung integrieren, die den Polizeibeamten den Weg zum Ziel weist. Analog zum Polizeiszenario könnte ein Spediteur seine LKW-Flotte überwachen und jederzeit von seinem Schreibtisch den Aufenthaltsort seiner Fahrzeuge betrachten, um zu überprüfen, wer wo ist und ob das Ziel rechtzeitig erreicht werden kann und ggf. darauf reagieren. Auf dem Gebiet der Zoologie könnte man einzelne Tiere mit einem Sender ausstatten und so die Wanderungen des Tieres oder der Herde mitverfolgen.

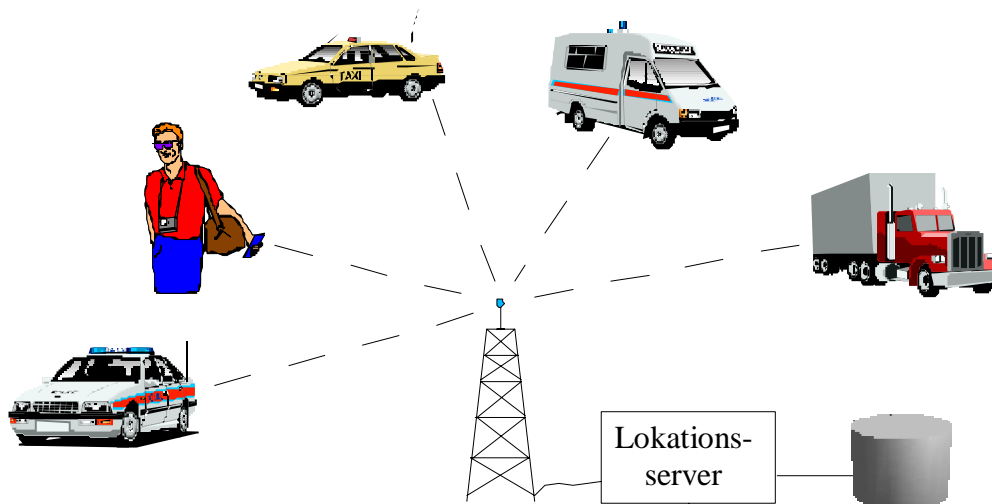


Abbildung 1 – Überblick über verschiedene Einsatzbeispiele, bei denen es von Bedeutung sein kann, den Aufenthaltsort zu kennen.

Um ortsbezogene Dienste anbieten zu können, müssen die Mobilobjekte ihre Positionen

an einen Server melden. Naturgemäß erfolgt diese Übermittlung über einen drahtlosen Kommunikationsweg, um die Mobilität zu gewährleisten. Drahtlose Nachrichtenübertragung zeichnet sich aber zum einen dadurch aus, daß die Bandbreite gering und die Kommunikation teuer ist. Wegen dieser beiden Gründe ist es erstrebenswert, möglichst wenige Daten übertragen zu müssen, um Kosten und Ressourcen einzusparen.

Ziel dieser Arbeit ist es, Protokolle für ortsbezogene Dienste zu entwickeln und zu bewerten, mit denen Updatenachrichten zur Mitteilung des momentanen Aufenthaltsortes eines Mobilien Objektes an den Server eingespart werden können. Dazu werden sogenannte Koppelnavigationsprotokolle (englisch: Dead-Reckoning-Protokolle) eingesetzt, die versuchen von einer Updatenachricht und dem bis dahin beobachteten Verhalten (Richtung, Geschwindigkeit) des Objektes ausgehend, die zukünftige Bewegung des Mobilien Objektes vorherzusagen. Weicht die Prognose von der tatsächlichen Position um weniger als ein festgelegter Toleranzradius ab, muß das Mobile Objekt keine neue Updatenachricht versenden. Erst wenn diese Abweichung größer als die Toleranzschwelle ist, verschickt der Client eine Nachricht an den Server. Der Server weiß also mit einer maximalen Ungenauigkeit in Höhe des Toleranzradius zu jedem Zeitpunkt, wo sich das betreffende Mobile Objekt befindet.

Die erzielten Resultate, die durch Simulation der Verfahren bestimmt wurden, zeigen sehr deutlich, daß der Einsatz von Koppelnavigation lohnenswert ist. Bereits bei einem sehr einfachen Verfahren wie dem Linearen Fortsetzen werden Einsparungen von bis zu 83% an Kommunikation gegenüber einem Nicht-Dead-Reckoning-Verfahren eingespart. Das komplexere Verfahren mit Kartenabgleich (Map Matching) erreicht sogar bis zu 91% Reduktion! (siehe Kapitel 7)

Die Arbeit findet im Rahmen des Projektes NEXUS an der Universität Stuttgart statt [NEXUS]. Das Projektziel ist, Konzepte und Verfahren zu erforschen, die für die Unterstützung von ortsbezogenen Diensten erforderlich sind. Das Resultat ist die Entwicklung einer Plattform für derartige Anwendungen. Beteiligt sind die Abteilungen Verteilte Systeme und Anwendungssoftware des Instituts für Verteilte Höchstleistungsrechner (IPVR), sowie das Institut für Photogrammetrie (IfP) und das Institut für Nachrichtentechnik und Datenverarbeitung (IND).

Im folgenden Kapitel werden die Grundlagen eingeführt. Dabei geht es insbesondere um die Fragen: Was ist Koppelnavigation und wie kann man diese Verfahren unterteilen? Anschließend werden im Kapitel 3 die verschiedenen Verfahren näher betrachtet und analysiert. Zum einen werden sie verglichen und ein Ausblick auf die zu erwarteten Simulationsergebnisse gegeben. Zum anderen werden ein paar wichtige Punkte besprochen, die bei einer Implementierung zu beachten sind. Im folgenden Kapitel 4 werden die Gemeinsamkeiten im Hinblick auf die Implementierung der Verfahren dargestellt. Zudem wird ein Kommunikationsprotokoll skizziert, das für das Funktionieren des Verbindungsaufbaus und -abbaus sowie der eigentlichen Datenübertragung erforderlich ist.

Anschließend werden die beiden Verfahren Lineares Fortsetzen und Kartenbasierte Koppelnavigation genauer betrachtet, da diese für die Simulation implementiert wurden. Dabei liegt das Hauptaugenmerk auf den Details, die für eine praktische Umsetzung zu beachten sind. Schließlich wird die Simulation selbst beschrieben und die Ergebnisse der Arbeit präsentiert.

## 2 Grundlagen

---

Das Grundlagenkapitel soll in den Themenkomplex dieser Arbeit einführen. Auf die an dieser Stelle präsentierten Grundlagen wird in den später folgenden Kapiteln aufgebaut. Die hier eingeführten Begriffe werden immer wieder auftauchen.

### 2.1 Koppelnavigationsprotokolle

Die Koppelnavigation (englisch: dead reckoning) hat ihren Ursprung in der Schifffahrt. Bevor die Navigation mittels der Gestirne im 15. Jahrhundert entwickelt wurde, war es die einzige Möglichkeit, die Position zu bestimmen. Bei der Koppelnavigation wird die momentane Position anhand des Kurses und der Entfernung der zuletzt bestimmten Position bestimmt. Ausgehend von einem bekannten Startpunkt trägt dabei der Navigator Kurs und Entfernung in der Karte ein. Die zurückgelegte Entfernung  $s$  wird durch eine einfache Berechnung aus der verstrichenen Zeit  $t$  und der Geschwindigkeit  $v$  bestimmt ( $s=v \cdot t$ ).

Heutzutage wird die Koppelnavigation auch im Flugverkehr verwendet (auch wenn zunehmend andere Systeme wie GPS eingesetzt werden). Die Flugzeuge richten ihre Wege nach sogenannten Funkfeuern aus. Das sind ortsfeste Funkstationen, deren Position bekannt ist und die auf festgelegten Frequenzen ein Signal aussenden. Überfliegt ein Flugzeug ein Funkfeuer, weiß es, an welchem Ort es sich befindet. Zwischen den Funkfeuern wird Koppelnavigation angewendet, um jederzeit die aktuelle Flugposition zu berechnen.

In der hier vorliegenden Arbeit kommt Koppelnavigation etwas anders als in den obigen Beispielen zur Anwendung. Das Mobile Objekt verfügt über ein Lagemeßsystem, welches die Fähigkeit besitzt, absolute Positionen zu bestimmen (auf welche Art auch immer). Das ist ein wichtiger Unterschied zu den obigen Beispielen bei Schiffen oder Flugzeugen. Trotzdem wird hier Koppelnavigation eingesetzt, um ausgehend von einer absolut bestimmten Position nachfolgende Lagen zu berechnen.

Die Vorgehensweise ist dabei wie folgt. Zunächst wird eine Ist-Position bestimmt. Das ist, wie bereits erwähnt, die Aufgabe des Sensorsystems des Mobilen Objektes. Diese Ist-Position wird vom Mobilen Objekt an den Server über einen drahtlosen Kommunikationskanal gesendet. Nun ist auch dem Lokationsserver die absolute Position des Objektes zum Zeitpunkt des Abschickens der Nachricht bekannt<sup>1</sup>. Neben der Positionsinformationen werden aber auch Informationen über Geschwindigkeit und Bewegungsrichtung übermittelt. Anhand dieser Daten ist der Server jederzeit in der Lage, zukünftige Positionen des Mobilen Objektes vorherzusagen (durch Koppelnavigation), wenn es sich mit der gleichen Charakteristik weiterbewegt. Insbesondere ist er in der Lage, Anfragen dritter nach dem Aufenthaltsort des betreffenden Objektes ohne Rückfragen beim Objekt zu beantworten.

---

<sup>1</sup> Man muß beachten, daß die Kommunikation eine gewisse Zeit in Anspruch nimmt! Daher ist die Positionsmeldung nicht mehr ganz aktuell, weil sich das Objekt weiterbewegt haben könnte.



kann es erforderlich werden, eine Updatenachricht zu verschicken. Dies ist beispielsweise wegen eines Haltes an einer roten Ampel der Fall. Unter einer falschen Wegprognose ist zu verstehen, daß der Koppelnavigationsalgorithmus einen anderen Weg berechnet als das Mobile Objekt tatsächlich genommen hat. Beispielsweise kann es daran liegen, daß das Mobile Objekt eine Abzweigung nimmt, während der Algorithmus davon ausgeht, daß sich das Objekt weiter geradeaus bewegt.

Es gibt verschiedene Koppelnavigationsverfahren. Sie werden im nachfolgenden Kapitel vorgestellt. Die Art, wie die Soll-Lagen berechnet werden, und welche Modelle zur Vorhersage der Bewegung dahinter stehen, unterscheiden die Verfahren.

Aus der Sicht des Lokationsservers ist die Vorgehensweise im Prinzip mit den Funkfeuern der Luftfahrt vergleichbar. Immer wenn ein Funkfeuer überflogen wird, kennt das Flugzeug die genaue Position. Anschließend wird Koppelnavigation angewendet, um die Zwischenpositionen zu bestimmen. Wenn das nächste Funkfeuer erreicht ist, ist die Lage wieder mit größter Genauigkeit bekannt. Bei den Mobilien Objekten ist der Empfang einer Updatenachricht vom Server mit dem Überflug über ein Funkfeuer vergleichbar. Der Server kennt die genaue Position des Objektes. Anschließend wird Koppelnavigation eingesetzt um Zwischenpositionen zu bestimmen. Im Unterschied zum Funkfeuer-Beispiel verschickt aber das Mobile Objekt seine Updates nicht nach Zurücklegung einer festen Entfernung, sondern nur dann, wenn die Soll-Lage der Koppelnavigation mehr als ein tolerierbares Maß von der Ist-Position abweicht.

Zu beachten ist, daß das Mobile Objekt zur Bestimmung seiner momentanen Position *nicht* die Hilfe der Koppelnavigation benötigt! Nur Soll-Positionen werden durch Dead-Reckoning berechnet, auf die der Server angewiesen. Um aber vergleichen zu können, ob Soll- und Ist-Lagen nicht zu weit voneinander abweichen, muß auch das Mobile Objekt Koppelnavigation betreiben, um die Soll-Positionen zu berechnen.

Ziel des Einsatzes eines Koppelnavigationsverfahrens ist es, die Anzahl der Updatenachrichten zu minimieren, da Kommunikation über Funkkanäle teuer ist. Außerdem ist die Bandbreite bei mobiler Kommunikation gering. Trotzdem soll der Server aber jederzeit in der Lage sein, Anfragen nach dem Aufenthaltsort eines bestimmten Objektes mit einer Genauigkeit zu beantworten, die für die jeweilige Anwendung hinreichend groß sind. Die Genauigkeit wird über die Festlegung des Toleranzradius  $d_{max}$  bestimmt. Im nachfolgenden Kapitel wird eine Übersicht der verschiedenen Koppelnavigationsprotokolle gegeben, die man unterscheiden kann. Der prinzipielle Ablauf ist aber stets wie hier beschrieben. Lediglich der Aufwand für die einzelnen Verfahren ist unterschiedlich. Entsprechend sind auch bessere oder weniger gute Resultate bei der Anzahl der zu verschickenden Updatenachrichten zu erwarten.

## **2.2 Taxonomie der Verfahren**

Man kann mehrere Verfahren unterscheiden. In Abbildung 3 sind die unterschiedlichen Verfahren dargestellt, in die man die Koppelnavigationsprotokolle zur Einsparung von Kommunikationsaufwand zwischen dem Mobilien Objekt und dem Lokationsserver einteilen kann. Die Unterscheidung erfolgt anhand der Methode, die zur Vorhersage der Positionen verwendet wird.

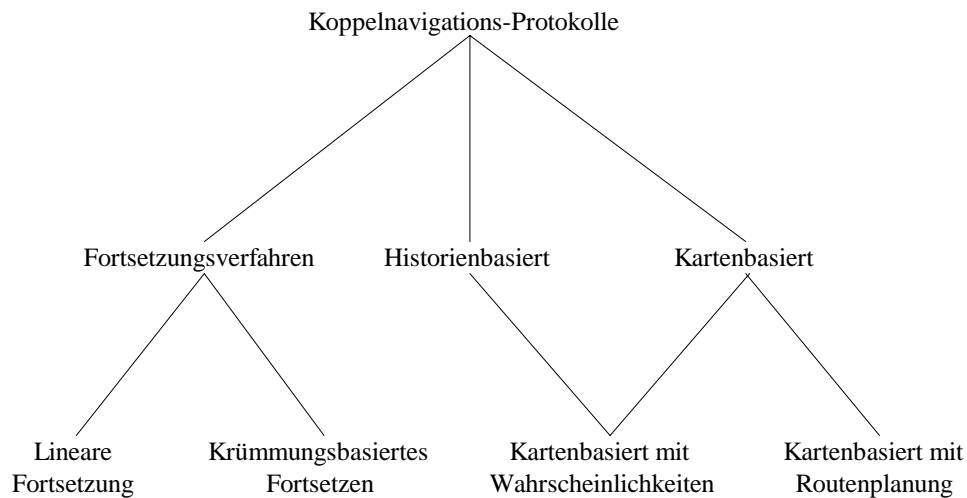


Abbildung 3 - Taxonomie der Koppelnavigationsverfahren

Nachfolgend werden die in der Abbildung dargestellten Verfahren einzeln beschrieben.

## 2.2.1 Fortsetzungsverfahren

Bei den Fortsetzungsverfahren werden folgende Informationen zur Vorhersage der Soll-Positionen (d.h. die zukünftigen Aufenthaltsorte) benötigt:

- *aktuelle Ist-Position*  
vom Lagemeßsystem bestimmt. Sie dient als Startpunkt der Koppelnavigation
- *zuletzt ermittelte Ist-Positionen*  
zur Berechnung der Bewegungsrichtung und ggf. der (Kurven-)Geometrie
- *Geschwindigkeit der Bewegung*

Zur Funktion der Verfahren sind darüber hinaus keine Zusatzdaten (wie elektronische Karten) nötig wie sie bei den Verfahren späterer Kapitel benötigt werden.

Wie bei der klassischen Koppelnavigation in der Schifffahrt wird angenommen, daß sich das Mobile Objekt in Bewegungsgeometrie und -geschwindigkeit weiter gleich. Daraus werden die zukünftige Positionen berechnet.

Man kann zwei Fortsetzungsverfahren unterscheiden: Lineare Fortsetzung und Krümmungsapproximation. Diese sollen im folgenden näher erläutert werden.

### 2.2.1.1 Lineare Fortsetzung

Bei der linearen Fortsetzung handelt es sich um ein Fortsetzungsverfahren, das stets davon ausgeht, daß sich das Mobile Objekt von der gegenwärtigen Position gerade (also linear) in die gleiche Richtung und mit gleicher Geschwindigkeit weiterbewegt wie bisher.

Ist das Lagemeßsystem in der Lage, eine Orientierung zu bestimmen, kann diese Orientierung als Richtungsvektor für die weitere Bewegung benutzt werden. Ansonsten muß die Richtung aus den gemessenen Ist-Positionen errechnet werden. Im einfachsten Fall wird der

Richtungsvektor aus der momentanen Position und der zuletzt gemessenen Position errechnet. Auf diese Weise kann man auch die Geschwindigkeit berechnen, ohne einen speziellen Geschwindigkeitsmesser einzusetzen.

Um die Ungenauigkeiten des Lagemeßsystems möglichst zu minimieren, kann man die Bewegungsrichtung nicht nur aus einem Meßpunkt bestimmen, sondern aus mehreren. Aus den Ist-Positionen bestimmt man dazu eine Regressionsgerade, die als Bewegungsrichtung dient. Näheres dazu wird später im Kapitel 5.1 erläutert.

### **2.2.1.2 Krümmungsapproximation**

Im Unterschied zur linearen Fortsetzung geht die Krümmungsapproximation nicht davon aus, daß die Bewegung stets linear weitergeht. Statt dessen wird aus den letzten Ist-Positionen eine Kurve errechnet, von der angenommen wird, daß sie der zukünftigen Bewegung des Mobilobjektes entspricht. Beiden Fortsetzungsverfahren ist aber gemeinsam, daß die Bewegungsgeschwindigkeit als konstant angenommen wird.

Bei diesem Verfahren ergeben sich mehr Freiheiten bei der Umsetzung als bei der linearen Fortsetzung. Man kann zur Fortsetzung sehr viele verschiedene Kurven einsetzen, um die Bewegung zu modellieren: Kreisbögen, Ellipsen, Bézier-Kurven, diverse Splines, usw. Man kann approximierende und interpolierende Verfahren<sup>2</sup> einsetzen.

## **2.2.2 Kartenbasierte Verfahren**

Kartenbasierte Verfahren setzen – wie der Name schon andeutet – Karten ein, um Update-Nachrichten einzusparen. Im einfachsten Fall wird lediglich eine Karte benutzt. Positionen werden dabei auf die Kanten der Karte abgebildet (Map Matching). Es wird angenommen, daß sich das Mobile Objekt weiter entlang dieser Kanten bewegt. Verfügt man über die Information, welches Ziel erreicht werden soll, kann man das Verfahren anwenden, das eine Routenplanung einschließt. Der letzte Fall schließlich setzt neben einer Karte auch noch Historien ein, um Wahrscheinlichkeiten für die Wahl eines Weges verwenden.

Nachfolgend werden diese drei Verfahren beschrieben.

### **2.2.2.1 Einfaches kartenbasiertes Verfahren**

Bei diesem Verfahren, haben das Mobile Objekt und der Lokationsserver elektronische Kartendaten, die sie zur Durchführung der Koppelnavigation einsetzen. Die Karten werden dabei als Graph repräsentiert. Die Kanten des Graphs stellen Straßen dar. Die Knoten sind Start- und Endpunkte der Kanten. Die Repräsentation der Daten wird in einem späteren Kapitel vertieft (Kapitel 6.1, Seite 55). Zur Repräsentation von Karten existieren Standards wie GDF (Geographic Data File) oder ATKIS (Amtlich Topographisch-Kartographisches Informationssystem). Mehr dazu im Kapitel 6.1.3.

Die vom Lagemeßsystem gemeldeten Positionen werden auf eine Kante der Karte gematcht. Der Vorteil dieses Verfahrens ist, daß Kurven im Voraus bekannt sind, so daß im günstigsten

<sup>2</sup> Bei einem approximierenden Verfahren geht die berechnete Kurve durch die vorgegebenen Stützpunkte (hier die Ist-Positionen). Bei einem interpolierenden Verfahren gehen die Kurvenpunkte nicht zwangsläufig durch die Stützpunkte. Die Stützpunkte beeinflussen zwar das Aussehen der Kurve, liegen aber nicht zwingend auf der Kurve selbst.

Fall der Richtungswechsel beim Übergang von einem Geradenstück auf eine Kurve (und später wieder zurück) zu keiner Update-Nachricht führt.

### 2.2.2.2 Routenplanung

Für die Verwendung der Routenplanung werden neben den Kartendaten auch noch Informationen verwendet, welches Ziel das Mobile Objekt erreichen möchte. Sie ist in der Autonavigation weit verbreitet. Ausgehend von der aktuellen Position wird dann die günstigste Route zum Ziel berechnet. Die günstigste Route kann je nach Optimierungsziel anders ausfallen: Es kann die schnellste, kürzeste oder nach anderen Gesichtspunkten optimalste Strecke sein.

Anhand der Routenplanung bei der Koppelnavigation ist es Client und Server bekannt, welche Ausgangskanten an Kreuzungspunkten gewählt werden. Dadurch ist eine weitere Einsparung an Update-Nachrichten gegenüber dem reinen kartenbasierten Fall zu erwarten.

Eine Untersuchung eines derartigen Verfahrens ist unter [WOL99] nachzulesen.

### 2.2.2.3 Kartenbasiert mit Wahrscheinlichkeiten

Dieser Fall ist eigentlich eine Mischform aus dem historienbasierten Verfahren (siehe unten) und dem kartenbasierten Ansatz. Die Kanten der Karte erhalten Wahrscheinlichkeitswerte zugewiesen, die relative Häufigkeiten ausdrücken, mit denen eine bestimmte Ausgangskante gewählt wird. Die Wahrscheinlichkeitswerte gewinnt man aus der Untersuchung von Historien.

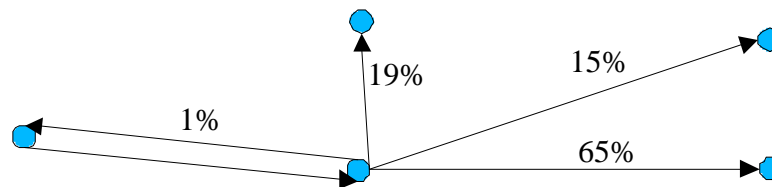


Abbildung 4 - Beispiel für die Wahrscheinlichkeitsverteilung an einem Kreuzungspunkt.

Wie beim Historienbasierten Verfahren kann man die Wahrscheinlichkeiten global (d.h. für alle Mobilen Objekte gültig) berechnen, die Mobilen Objekte in Klassen unterteilen oder für jedes Mobile Objekt eine eigene Wahrscheinlichkeitsverteilung benutzen.

Natürlich ist mit diesen Wahrscheinlichkeiten keine sichere Vorhersage zu treffen, welche Kante gewählt wird! Nur wenn sich das Mobile Objekt so verhält wie die meisten an diesem Punkt, werden Updates eingespart.

## 2.2.3 Historienbasierte Koppelnavigation

Bevor historienbasierte Koppelnavigation angewandt werden kann, müssen zunächst Daten gesammelt werden: die Historien. Eine Historie besteht aus dem Protokoll der Bewegung eines Objektes in einem bestimmten Bereich. Hat man viele Historien gesammelt, versucht man Gemeinsamkeiten zu finden. Das entspricht einer Art Data Mining auf den Historien, um

Gemeinsamkeiten zu bestimmen. Die Grundannahme bei der historienbasierten Koppelnavigation ist, daß immer wiederkehrende Wegstrecken in den einzelnen Historien vorkommen. Die häufigsten Wege, die aus den Historien durch Data Mining destilliert wurden, ergeben eine Art Karte. Die Wahrscheinlichkeit, daß sich ein Mobiles Objekt entlang einer Kartenkante bewegt, ist relativ hoch, wenn viele Mobile Objekte in ihren Historien die gleiche Kante enthielten. Man kann die Historien entweder allgemein, nach Klassen getrennt oder für einen spezifischen Benutzer aufnehmen. Das bedeutet, daß man entweder für alle Mobilen Objekte die selbe Karte benutzt oder für jede Klasse bzw. für jeden Benutzer jeweils eine eigene Karte erzeugt.

Das eigentliche Koppelnavigationsprotokoll bedient sich dann später der so gewonnenen Karten. Wenn sich nun ein Mobiles Objekt auf einer Kante befindet, wird angenommen, daß es sich weiter entlang der Kante bewegt. Ist diese Vorhersage richtig, greift das Protokoll und spart so Update-Nachrichten zwischen Client und Server ein. Im Gegensatz zum obigen vorgestellten kartenbasierten Ansatz wird aber nicht angenommen, daß sich das Objekt entlang einer Kante bewegen muß. Daher erfolgt auch kein Map Matching zur nächstgelegenen Kante. Eine Kante beim historienbasierten Ansatz stellt nur eine statistisch bevorzugte Bewegungskante dar. Der kartenbasierte Ansatz geht an dieser Stelle weiter und geht davon aus, daß die Kanten die einzig möglichen Bewegungswege darstellen.

Hat man sehr viele Historien aufgenommen, bekommt man eine sehr gute Karte, von der man annehmen kann, daß alle möglichen Wege enthalten sind. Bei dieser strikten Auslegung des Historienbasierten Verfahrens unterscheidet sich das Verfahren nicht mehr vom Kartenbasierten Verfahren.

## **2.3 Relevante Koordinatensysteme**

In der vorliegenden Arbeit werden zwei Klassen von Koordinatensysteme vorkommen. Das ist zum einen ein geographisches Koordinatensystem, bei dem die Positionen in Längen- und Breitengraden angegeben werden. Zum anderen kommen karthetische Koordinatensysteme vor, bei denen die Koordinatenachsen senkrecht aufeinander stehen.

Das geographische Koordinatensystem WGS84 (World Geodetic System 1984) wird vom GPS (Global Positioning System) benutzt. Ein GPS-Sensor – wie wir ihn auch für unsere Simulation im Einsatz hatten – liefert also Positionswerte in diesem Koordinatensystem. Das sind die geographischen Koordinaten Breite, Länge und die ellipsoidische Höhe. Näheres dazu ist unter [FRÖ95] und [BAU97] nachzulesen.

Zur Minimierung der Fehler bei der Berechnung von Richtungen und Entfernungen aus den Positionsdaten, wie sie bei unseren Koppelnavigationsprotokollen benötigt werden, wird ein karthetisches Koordinatensystem benötigt. WGS84 erfüllt diese Bedingung nicht. Daher bedarf es einer Überführung durch eine Transformation in ein rechtwinkliges Koordinatensystem [HEC95]. Zwei solche Koordinatensysteme sind UTM (Universal Transversal Mercator) und Gauß-Krüger.

Das Gauß-Krüger-System [BAU97], das im Rahmen der Arbeit verwendet wurde, ist eine winkeltreue Abbildung von Punkten in ein ebenes rechtwinkliges Koordinatensystem. Es wird oft für Karten verwendet. In der Bundesrepublik Deutschland werden 3° breite

Meridianstreifen verwendet. Die Zählung der Hauptmeridiane geht dabei vom Meridian von Greenwich aus. Die einzelnen Streifen sind Kennziffern zugeordnet, die sich aus der nach Osten positiv gezählten geographischen Länge der Hauptmeridiane ergeben. Das Gebiet der BRD wird durch die Meridianstreifensysteme mit den Kennziffern 2,3 und 4 überdeckt.

Zu den y-Werten der Koordinate werden 500000 und der Wert  $(\text{Hauptmeridian}/3^\circ) \cdot 1000000$  addiert. Die Angabe 4313360 bedeutet also, daß der Ort 186,640km westlich vom Meridian  $12^\circ$  liegt. Die x-Werte drücken die Entfernung zum Äquator aus. (Werte auf der südlichen Halbkugel sind unüblich.) So bedeutet 5903137, daß der Punkt 5903,137km nördlich des Äquators liegt. Wie man dem Beispiel entnehmen kann, sind die Koordinatenwerte jeweils in Metern zu interpretieren.

## 3 Diskussion der Protokolle

---

In diesem Kapitel geht es im ersten Teil darum, die im Grundlagenkapitel vorgestellten Verfahren zu diskutieren und dabei deren jeweilige Vor- und Nachteile zu beleuchten. Diese Diskussion ist statisch, d.h. sie berücksichtigt die theoretischen Eigenschaften der Verfahren. Die Resultate, die sich durch Implementierung und Simulation ergeben sind dagegen im Kapitel 7 auf Seite 80 nachzulesen.

Im zweiten Teil dieses Kapitels werden dann noch ein paar Sachverhalte besprochen, die bei einer konkreten Implementierung der Verfahren beachtet werden müssen. Dabei werden die Probleme aufgezeigt und Lösungsvorschläge geboten.

### 3.1 Bewertung der Verfahren

Nachfolgend sollen die im Grundlagenkapitel vorgestellten Verfahren noch einmal aufgegriffen und hinsichtlich ihrer Eigenschaften gegenübergestellt werden. Dabei geht es um eine Bewertung der Vor- und Nachteile der jeweiligen Verfahren.

Sofern Hardwareanforderungen besprochen werden, beziehen sie sich auf die Client-Seite, weil hier die größeren Restriktionen bestehen. Der Computer des Mobilobjektes unterliegt (in den meisten Fällen) der starken Einschränkung durch die Batterie bzw. der Akkulaufzeit. Abgesehen von Kraftfahrzeugen, die durch den Motor, den Strom selbst herstellen können, muß ein mobiles Objekt eine Batterie bzw. Akku als Stromversorgung haben. Mit der heutigen Technologie sind aber diese Energiespeicher sehr schwer und klobig. Es ist also darauf zu achten, daß der Betrieb der Client-Geräte möglichst wenig Strom benötigt, um die Akkus klein zu halten. Weitere Parameter sind das Gewicht und das Volumen (also die Größe) der Geräte. Insbesondere bei Geräten, die von Personen zu tragen sind, ist darauf zu achten, daß das Gerät möglichst klein und leicht ist.

Für den Server sind diese Kriterien von geringerer Bedeutung, auch wenn sie sich sinngemäß übertragen lassen. Bei einem Lokationserver laufen die Daten von etlichen mobilen Objekten zusammen. Seine Rechenkapazität muß ausreichen, um seinerseits die Soll-Positionen aller verwalteten Objekte periodisch neu zu berechnen. Je komplexer ein Algorithmus ist, desto weniger mobile Objekte kann ein Server bei gleicher Prozessorleistung betreuen. Alternativ muß die Prozessorleistung erhöht werden, um die gleiche Anzahl an Clients pro Server zu verarbeiten. Der Stromverbrauch ist zwar bei einem stationären Gerät, wie es ein Server darstellt, zweitrangig, aber mehr Stromverbrauch bedeutet mehr Energiekosten und mehr Kühlung, was bei modernen Prozessoren ein immer größeres Problem wird, weil die Verlustleistungen mit zunehmenden Taktfrequenzen steigen! Für den Server ist also die größte Restriktion die Komplexität des eingesetzten Algorithmus.

Neben den Hardwareanforderungen für Server und Clients darf man aber auch das eigentliche Optimierungsziel dieser Arbeit nicht außer Acht lassen: die Kommunikation zwischen den

beiden auf ein Minimum zu beschränken. Tendenziell ist zu erwarten, daß komplexere Verfahren weniger Updates benötigten als die einfachen Algorithmen. Diesen Vorteil erkaufte man sich aber durch den Nachteil, daß die Hardwareanforderungen mit der Komplexität des Verfahrens steigen.

Nachfolgend werden die Verfahren analog zur Taxonomie diskutiert, die im Kapitel 2.2 vorgestellt wurde: Dabei werden zunächst die Fortsetzungsverfahren, dann die historienbasierte Koppelnavigation und schließlich die kartenbasierten Verfahren dargestellt. Die hierbei vorgestellten Ergebnisse fußen allerdings nicht auf einer Simulation, sondern stellen eine theoretische Betrachtung dar. Eine Simulation und die erhaltenen Simulationsergebnisse werden erst in einem späteren Kapitel wiedergegeben (Kapitel 7, Seite 80).

### 3.1.1 Fortsetzungsverfahren

Die Fortsetzungsverfahren sind die einfachsten Verfahren. Das gilt sowohl hinsichtlich der Hardwareanforderungen als auch der Komplexität der Implementierung. Die lineare Fortsetzung hat von allen Verfahren den geringsten Bedarf an Rechenkapazität. Zur Berechnung der Soll-Positionen sind lediglich ein paar wenige simple Berechnungen notwendig. Anhand der Updatenachricht ist bereits die Bewegungsrichtung und die -geschwindigkeit bekannt. Durch die verstrichene Zeitdifferenz zwischen der letzten Position und der aktuellen Zeit, läßt sich die zurückgelegte Strecke berechnen. Dadurch ist es durch einfache Addition eines Vektors zu einem Punkt möglich, die aktuelle Soll-Position zu bestimmen. Bei der Krümmungsapproximation ist das Arbeitsprinzip ähnlich. Der einzige Unterschied ist der, daß nicht nur linear fortgesetzt wird, sondern eine beliebige Kurve für den Bewegungsverlauf angenommen wird. Nach der Berechnung der zurückgelegten Strecke ist es bei einer beliebigen Kurve etwas komplexer, den Kurvenpunkt zu bestimmen, der von der Ausgangsposition die berechnete Bogenlänge besitzt. Daher wird beim kurvenbasierten Verfahren der Prozessor stärker gefordert.

Beim kurvenbasierten Verfahren muß auch zur Berechnung der Updates mehr Rechenaufwand investiert werden. Aus den letzten Ist-Positionen muß eine geeignete Kurve approximiert oder interpoliert werden. Dazu sind deutlich mehr Rechenschritte notwendig als nur aus den letzten beiden Lagepunkten eine Richtung zu errechnen. Wird beim linearen Fortsetzungsverfahren dagegen eine Regressionsgerade berechnet, um die Bewegungsrichtung aus den letzten Lagepunkten zu mitteln, wird auch bei diesem Verfahren ein mit dem kurvenbasierten Verfahren vergleichbarer Rechenaufwand benötigt.

Die Updates des Kurvenapproximierenden Verfahrens müssen entweder alle Punkte enthalten, aus denen die Kurve errechnet werden kann, oder in einer geeigneten Art die Funktion der Kurve speichern. Bei der Linearen Fortsetzung reicht die Angabe der Bewegungsrichtung. Folglich sind die Updates beim Kurvenverfahren größer, da sie mehr Daten enthalten.

Ein Vorteil ist, daß sowohl das lineare Fortsetzen als auch die Krümmungsapproximation nicht sehr schwierig zu implementieren sind. Allerdings sind die Algorithmen der Krümmungsapproximation etwas komplexer, weil beliebige Kurven mathematisch komplizierter zu behandeln sind als einfache Geraden.

Beiden Fortsetzungsverfahren ist gemeinsam, daß sie keine Zusatzdaten benötigen wie sie beispielsweise die Karten der kartenbasierten Ansätzen darstellen. Dadurch sind die Anforderungen an den Speicherbedarf recht bescheiden. Das ist insbesondere beim Client von Vorteil, da Einsparung von Speicher sowohl Kostenvorteile als auch Vorteile für den Betrieb mit sich bringt. Vorteile für den Betrieb sind Gewichtseinsparung, weniger Platzbedarf und weniger Stromverbrauch, da keine Zusatzhardware betrieben werden muß.

Die Fortsetzungsverfahren haben zwar die geringsten Anforderungen an die Hardware, dürften aber auf Grund des einfachen Algorithmus auch im Vergleich zu den anderen Verfahren die schlechtesten Update-Raten aufweisen. Dabei schneidet allerdings das lineare Fortsetzen etwas schlechter ab als die Krümmungsapproximation. Die Erwartung begründet sich darauf, daß die Fortsetzungsverfahren keinerlei Information über ihre Umwelt besitzen. Sie gehen stur immer davon aus, daß sich die Bewegung mit der gleichen Charakteristik fortsetzen wird wie sie aus dem historischen Verhalten bestimmt wurde. Das etwas bessere Abschneiden der Krümmungsapproximation ist damit zu erklären, daß nicht nur lineare Bewegungen erfaßt werden, sondern auch gekrümmte Bewegungen. Kurvenfahren eines Mobilien Objektes können so modelliert werden. Ist die Modellierung gelungen, können weitere Updates während einer Kurvenfahrt eingespart werden.

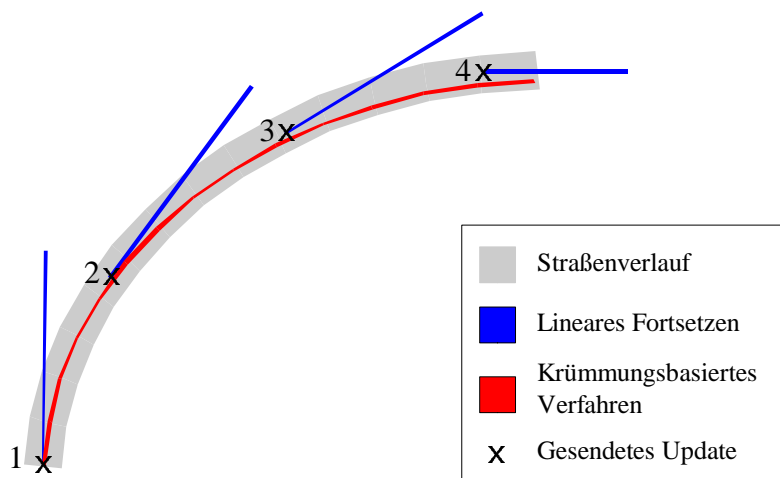


Abbildung 5 – Verdeutlichung des Unterschieds zwischen der Kurvenfahrt mit linearem Fortsetzen und Krümmungsapproximation. Obwohl die krümmungsapproximierende Kurve den Straßenverlauf nicht exakt modelliert, wird nur 1 Update gesendet. Beim linearen Fortsetzen sind es in diesem Beispiel dagegen 4 Updates.

### 3.1.2 Historienbasierte Koppelnavigation

Bei der historienbasierten Koppelnavigation wird aus den gesammelten Historien eine Karte gewonnen. Zweifellos werden zur Erzeugung einer solchen Karte sehr viele Rechenschritte benötigt. Das wiederum bedeutet, daß ein leistungsfähiger Prozessor benötigt wird, um diese Aufgabe zu bewerkstelligen. Allerdings ist dieser Aufwand einmalig, da die Karte statisch vor dem Start des Koppelnavigationsalgorithmus berechnet werden muß. Diese Aufgabe muß auch nicht zwingend vom Client-Gerät durchgeführt werden, sondern kann von einem leistungsfähigen Rechner (nicht zwingend der Server) erfolgen. Ist die Karte generiert, werden lediglich die fertigen Kartendaten auf das mobile Gerät überspielt.

Die sonstigen Hardwareanforderungen hinsichtlich der Speicherung der Kartendaten und die Prozessoranforderungen sind mit dem einfachen Fall des kartenbasierten Verfahren identisch. Daher werden sie auch an dieser Stelle nicht dargestellt.

Ein Unterschied zu den kartenbasierten Verfahren ist der, daß zwar angenommen wird, daß sich die Mobilen Objekte bevorzugt auf den durch die Historien gewonnenen Kartendaten bewegen, aber sie es nicht zwingend müssen. Beim historienbasierten Ansatz ist es den Mobilen Objekten erlaubt, beliebige Wege zu beschreiten<sup>3</sup>. Somit schwankt das Verfahren zwischen einem Fortsetzungsverfahren (wenn die Wege der Historien verlassen werden) und einem einfachen kartenbasierten Ansatz (wenn die Historienpfade beschritten werden). Zur Laufzeit muß immer wieder zwischen den beiden Verfahren umgeschaltet werden. Je nachdem, wie sich das Mobile Objekt verhält, liegen die Resultate näher beim einen oder beim anderen Extrem. Dementsprechend sind die zu erwarteten Einsparungen an Updates im Bereich zwischen den Fortsetzungsverfahren und dem einfachen Kartenverfahren angesiedelt.

Eine weitere Abgrenzung zum Kartenbasierten Verfahren ist, daß die Karten des Historienbasierten Algorithmus objektspezifisch sein können und nicht für alle Klassen von Mobilen Objekten gültig sein müssen.

### 3.1.3 Kartenbasierte Verfahren

Kartenbasierte Verfahren stellen im Gegensatz zu den Fortsetzungsverfahren die höchsten Ansprüche an die Hardware. Kennzeichnend für alle kartenbasierten Verfahren ist die hohe Menge an benötigtem Speicher, um die Kartendaten zu speichern. Der Speicherbedarf unterscheidet sich nur marginal zwischen den Verfahren. Dabei benötigt das kartenbasierte Verfahren mit Wahrscheinlichkeiten wegen der Zusatzinformation geringfügig mehr Speicherplatz als die anderen Verfahren, um die Wahrscheinlichkeiten zu jeder Kante zu speichern. Dieser Zusatzbedarf ist aber vernachlässigbar klein.

Zum Thema Prozessoranforderungen ist ebenfalls festzustellen, daß die kartenbasierten Verfahren mehr Rechenleistung erfordern als die Fortsetzungsverfahren. Die Verfahren benötigen zur Initialisierung – d.h. zum Abgleich der Startposition mit der Karte – viel Rechenleistung. Das liegt daran, daß die Kartendaten durchsucht werden müssen, um die Kartenkante zu finden, welche der aktuellen Position am besten entspricht. Danach ist jede neue Position mit der Karte abzugleichen, wobei die dabei zu berücksichtigenden Kanten sehr lokal sind. Zur Laufzeit des Algorithmus ist also die Prozessorleistung nicht so hoch wie bei der Initialisierung. Trotzdem muß der Prozessor pro Berechnungsintervall etwas mehr arbeiten als bei den Fortsetzungsverfahren. Der Grund dafür ist, daß den Kartenverfahren der lineare Fortsetzungsansatz zugrunde liegt. Für die Bewegung längs einer Kante wird angenommen, daß sich das Mobile Objekt mit gleichförmiger Geschwindigkeit in Richtung der Kante weiterbewegt. Zusätzlich zu dem Fortsetzungsverfahren muß jede neu bestimmte Ist-Position mit der Karte abgeglichen werden. Das ist ein Rechenaufwand, den die Fortsetzungsverfahren nicht haben. Dieser Zusatzaufwand begründet den etwas höheren Leistungsanspruch an den Prozessor.

---

<sup>3</sup> Prinzipiell kann man natürlich auch beim Kartenbasierten Ansatz die Objekte nicht zwingen, innerhalb des Kartenbereiches zu bleiben. Aber Primär handelt es sich dort um einen Ausnahmefall, wenn die Karte verlassen wird und nicht um einen Regelfall (wie hier), der dem Ansatz des Verfahren zu Grunde liegt!

Die obigen Anforderungen an den Prozessor gelten für alle kartenbasierten Verfahren. Zusätzlich dazu benötigt die Routenplanung (beim Verfahren mit integrierter Routenplanung) weitere Prozessorzeit. Allerdings ist dieser Zusatzaufwand im wesentlichen einmalig zur Festlegung der Route. Beim Übergang von einer Kante zur nächsten Kante muß lediglich geprüft werden, ob sie längs der Route liegt. Das erfordert aber keinen wesentlichen Zeitaufwand.

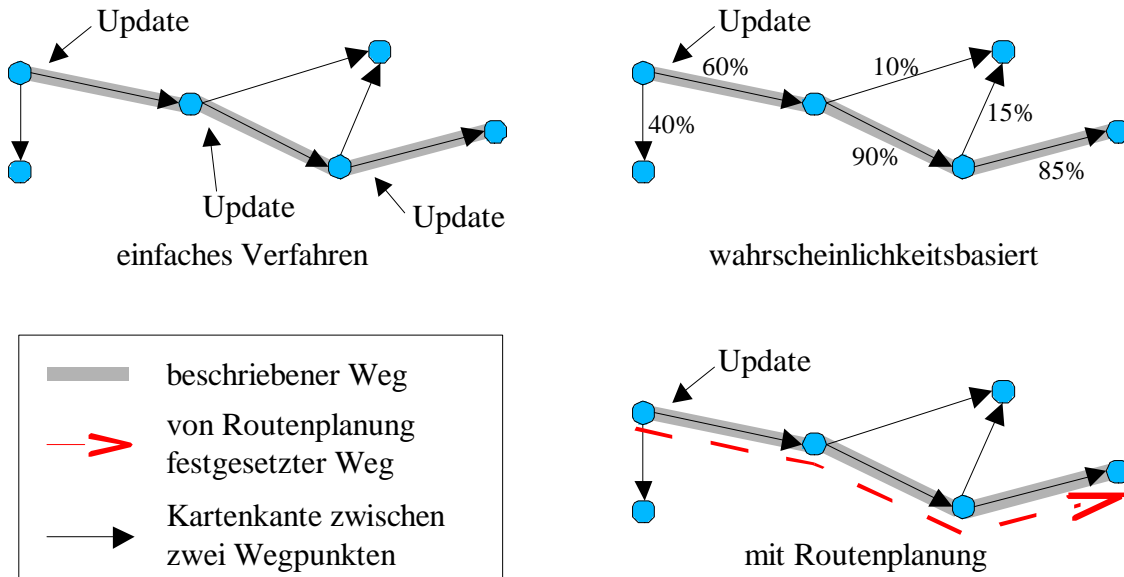


Abbildung 6 – Veranschaulichung der Unterschiede zwischen den drei kartenbasierten Verfahren bei der Updatepolitik nach einem Knotendurchlauf. Das einfache Verfahren muß immer dann ein Update schicken, wenn eine Kreuzung mit mehr als einer Ausgangskante durchlaufen wurde. Die beiden anderen Verfahren kommen mit jeweils einem Update aus, weil der genommene Weg der entsprechenden Vorhersage entspricht.

Die kartenbasierten Verfahren lassen auf Grund der Zusatzinformation, die sie in Form von Kartendaten verwenden, die besten Ergebnisse hinsichtlich der erwarteten Einsparung von Updatenachrichten erwarten. (Von den Historienbasierte Verfahren ist aber eine ähnliche oder gleiche Performance zu erwarten, wenn sehr viele Historien gesammelt wurden und der Bereich entsprechend vollständig abgedeckt ist.) Beliebige Streckenverläufe lassen sich ohne Sendung von Updatenachrichten durchfahren, sofern die Geschwindigkeit des Mobilen Objektes hinreichend konstant bleibt. Innerhalb der kartenbasierten Verfahren sind die Unterschiede gering. Die Unterschiede kommen durch die unterschiedliche Behandlung bzw. Behandlungsmöglichkeiten beim Übergang von einer Kartenkante zur nächsten zustande. Die Unterschiede treten zu Tage, wenn von einem Endknoten einer Kante mehrere Ausgangskanten ausgehen. Beim einfachen Verfahren werden alle Ausgangskanten gleich behandelt. Es muß zwangsläufig eine Updatenachricht gesendet werden, um den Server von der gewählten Ausgangskante zu unterrichten. (Zur Verbesserung der Performance kann man nach einem Knotendurchlauf diejenige Kante auswählen, welche zur bisherigen Kante den kleinsten Winkelunterschied aufweist. Dadurch werden die Resultate verbessert, ohne das Verfahren signifikant komplizierter zu machen.) Bei der Routenplanung steht die Ausgangskante a priori fest. Sofern das Mobile Objekt nicht die bestimmte Route verläßt,

muß keine Updatenachricht gesendet werden. Das gleiche Prinzip ist auch auf den Ansatz mit Wahrscheinlichkeiten zu übertragen. Die Ausgangskanten besitzen Wahrscheinlichkeiten, die eine Auskunft darüber geben, welcher Pfad vom Verkehr am häufigsten gewählt wird. Entspricht das Verhalten des Mobilobjektes der Statistik, dann wird auch in diesem Fall keine neue Updatenachricht benötigt. Die Unterschiede zwischen den Verfahren sind allerdings nur geringfügig, wenn angenommen wird, daß wenige Knoten mit mehreren Ausgangskanten durchlaufen werden oder die Kantenlängen sehr groß sind. Das gilt also vor allem für Überland- und Autobahnfahrten. Innerorts sollten dagegen die Einsparungen durch Routenplanung, aber auch durch den wahrscheinlichkeitsbasierten Ansatz, höher liegen.

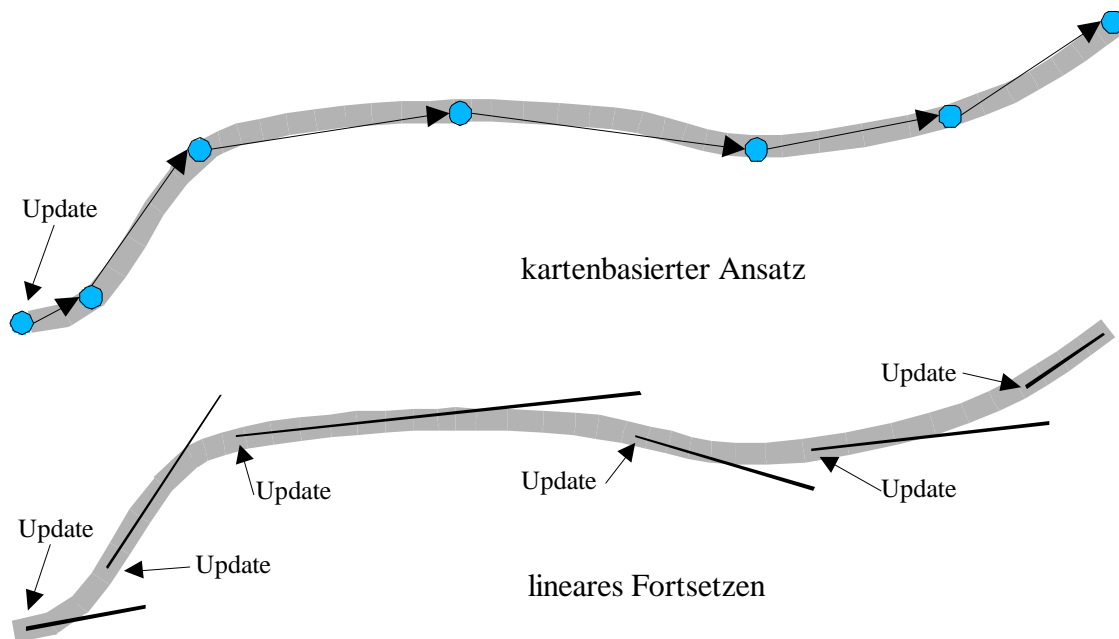


Abbildung 7 – Einsparung von Updates beim Durchschreiten einer kurvigen Straße beim kartenbasierten Ansatz im Vergleich zum linearen Fortsetzen. (Zur Vereinfachung wird im Beispiel angenommen, daß die Geschwindigkeit während der Kurvenfahrt konstant bleibt.)

Natürlich gilt auch hier für alle drei Verfahren, daß die Einsparung von Updates mit der Gleichförmigkeit der Geschwindigkeit des Mobilobjektes steht und fällt. Muß beispielsweise ein Auto an einer Ampel im Stadtverkehr halten, wird ein Update unabhängig vom Verfahren verschickt werden müssen.

Hinsichtlich der Implementierung ist zu den kartenbasierten Ansätzen zu sagen, daß die Verfahren die größte Komplexität aufweisen. Man muß mit der Karte abgleichen (Map Matching) und neue Kanten nach einem Knotendurchlauf suchen. Folglich wird eine effiziente Speicherung der Kartendaten benötigt, die den Zugriff erleichtern. Die Erfassung der Kartendaten selbst ist sehr zeitaufwendig und muß berücksichtigt werden. Alternativ kann man natürlich die Daten von einem Drittanbieter kaufen, was die beste Lösung darstellt.

### 3.1.4 Bewertung und Zusammenfassung

Für die Wahl eines Verfahrens gibt es keinen goldenen Weg. Man muß abwägen, ob man ein einfaches Verfahren mit geringen Hardwareanforderungen wie das lineare Fortsetzen wählt oder den größeren Aufwand für ein kartenbasiertes Verfahren eingeht. Dem Aufwand gegenüber stehen bei den komplexeren Algorithmen bessere Resultate, was die Einsparung an Updatenachrichten angeht.

Bereits das einfache Lineare Fortsetzen hat sich in der Simulation (siehe Kapitel 7) als sehr gut erwiesen. Im Vergleich zu einem Verfahren, das ohne Koppelnavigation auskommt, spart es bereits bis zu 83% der Updatenachrichten im günstigsten Fall! Der Kartenbasierte Ansatz kommt auf bis zu 91% Reduktion. Zum Fortsetzungsalgorithmus ergibt sich eine weitere Verbesserung von bis zu 60%.

Ausschlaggebend dürften wohl die Rahmenbedingungen sein, welche in der gewünschten Einsatzumgebung vorzufinden sind. Bei einem Fußgänger muß beispielsweise sehr stark auf ein niedriges Gewicht und Volumen geachtet werden. Der Computer darf die Tätigkeit nicht behindern oder nennenswert erschweren. Ist das Gerät zu groß oder zu schwer wird es von den Benutzern nicht akzeptiert werden!

Bei einem Gerät, das für den Autoeinbau gedacht ist, sind die konstruktiven Freiheiten größer. Das Gerät kann größer und schwerer ausfallen. Auch der Stromverbrauch tritt stärker in den Hintergrund, weil der laufende Motor über die Lichtmaschine Strom erzeugt.

Der letzte Komplex, der zu beachten ist, sind die Kosten für die Übertragung der Updates zum Server. Ist das Kommunikationsmedium sehr teuer, wird es wohl nötig sein, ein möglichst effizientes Verfahren ungeachtet der Hardwareanforderungen einzusetzen. Natürlich ist diese Entscheidung auch davon abhängig, welche Unterschiede zwischen den einzelnen Verfahren in der Praxis auftreten.

Eine Antwort auf die statistischen Unterschiede zwischen den Verfahren im tatsächlichen Einsatz liefert das Kapitel 7, ab Seite 80, in dem die Ergebnisse einer Simulation wiedergegeben werden.

Zum Abschluß hier noch eine tabellarische Übersicht der wichtigsten Eigenschaften der Verfahren:

	<i>Lineares Fortsetzen</i>	<i>Krümmungsapproximation</i>	<i>Historienbasiertes Verfahren</i>	<i>einfaches Kartenbasiertes Verfahren</i>	<i>Kartenbasiert mit Routenplanung</i>	<i>Kartenbasiert mit Wahrscheinlichkeiten</i>
leichte Umsetzbarkeit	5	3	1	1	1	1
Prozessoranforderung	1	2	4	4	5	4
Menge an Zusatzdaten beim Client	1 (keine Daten)	1 (keine Daten)	5	5	5	5

	<i>Lineares Fortsetzen</i>	<i>Krümmungsapproximation</i>	<i>Historienbasiertes Verfahren</i>	<i>einfaches Kartenbasiertes Verfahren</i>	<i>Kartenbasiert mit Routenplanung</i>	<i>Kartenbasiert mit Wahrscheinlichkeiten</i>
Update-Einsparungen	1	1 (aber etwas besser als Lineares Fortsetzen)	1 bis 5	5	5	5

**Zeichenerklärung:** 1 bedeutet *sehr wenig*, 5 steht für *sehr viel*

## 3.2 Diskussion der praktischen Umsetzung

Nachfolgend werden ein paar wichtige Probleme angesprochen, die bei der konkreten Umsetzung auftreten. Dabei werden auch mögliche Lösungswege aufgezeigt.

### 3.2.1 Einfluß von Rechenungenauigkeiten

Eine wichtige Voraussetzung für das Funktionieren von Dead-Reckoning ist, daß auf Servern und auf den Mobilien Geräten dieselben Vorhersagealgorithmen ablaufen. Da alle Dead-Reckoning-Verfahren darauf basieren, daß beide Seiten, Client und Server, ausgehend von einer Ist-Position eine Soll-Position berechnen, darf man die Rechendifferenzen nicht gänzlich außer Acht lassen. Solange sich aber die Resultate nicht zu sehr unterscheiden, führt es zwar zu leicht unterschiedlichen Ergebnissen, aber die Auswirkungen auf den Dead-Reckoning-Algorithmus sind nicht wesentlich.

Man kann in der Regel leider nicht davon ausgehen, daß Clients und Server identische Prozessoren einsetzen. Gründe dafür sind unterschiedliche Anforderungen an das jeweilige Gerät. Während ein Server viel Rechenleistung für seine Aufgaben benötigt und Kriterien wie Stromverbrauch, Kühlung und Platzbedarf eine untergeordnete Rolle spielen, sind es gerade diese Kriterien, die bei einem Mobilien Gerät sehr wichtig sind. Aus diesen unterschiedlichen Anforderungen ergeben sich zwangsläufig auch unterschiedliche Entscheidungen bei der Prozessorwahl. (siehe Kapitel 3.1)

Einzelne Prozessortypen können sich aber nicht nur in den erwähnten Merkmalen unterscheiden, sondern darüber hinaus auch in der Arithmetik. Allein schon nach dem Standard IEEE 754 gibt es Fließkommazahlen einfacher Genauigkeit (32 Bit) und doppelter Genauigkeit (64 Bit) [IEE85]. Neben diesem Standard gibt es auch Prozessoren, die proprietäre Formate einsetzen. Doch nicht nur die bloße Darstellung unterscheidet sich. Auch die Berechnungsalgorithmen für Division und trigonometrische Funktionen können sich unterscheiden, was ebenfalls zu leicht unterschiedlichen Ergebnissen führen kann [HAY88]. Gerade die trigonometrischen Funktionen werden für die Berechnung der Soll-Positionen sehr stark genutzt.

Verschiedene Genauigkeiten (und dadurch verschiedene binäre Darstellung der Zahlen) und sich unterscheidende Rechenalgorithmen führen aber zu leicht unterschiedlichen Rechenergebnissen.

In Abbildung 8 ist ein Beispiel zur Verdeutlichung der Auswirkung unterschiedlicher

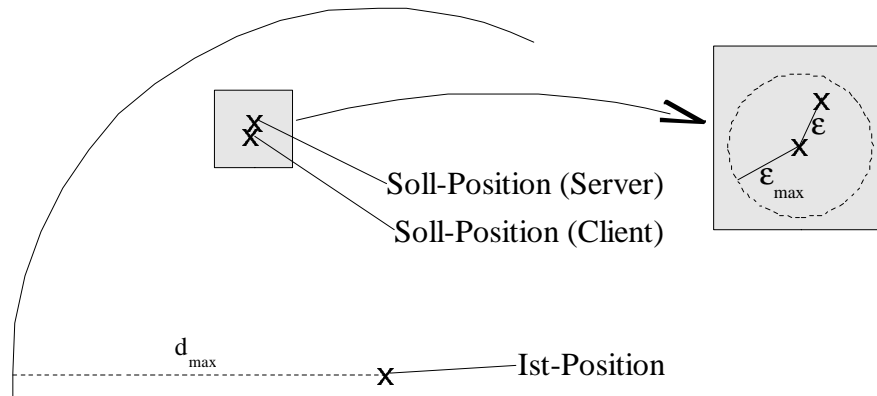


Abbildung 8 - Verdeutlichung der Auswirkung unterschiedlicher Rechengenauigkeit zwischen Client und Server. Auf der rechten Seite ist eine Vergrößerung abgebildet, welche die beiden Soll-Positionen und die Abstände  $\epsilon$  darstellt.

Rechengenauigkeiten bei Client und Server dargestellt. Abgebildet sind die vom Lagemeßsystem bestimmte Ist-Position (die nur dem Client bekannt ist) und die beiden durch den Algorithmus berechneten Soll-Positionen. Wie man erkennen kann, sind die Koordinaten der Soll-Positionen nicht gleich. Außerdem ist ein Teil des Kreises dargestellt, der die Punkte charakterisiert, die von der Ist-Position die Entfernung  $d_{max}$  haben<sup>4</sup>.

Auf der rechten Seite ist der Ausschnitt der beiden Soll-Positionen vergrößert dargestellt. Der Abstand zwischen diesen beiden Punkten wird als  $\epsilon$  bezeichnet. Zudem sei  $\epsilon_{max}$  der maximal auftretende Abstand zwischen diesen beiden Punkten. ( $\epsilon_{max}$  ist also das Maximum aller  $\epsilon$ :

$$\forall \epsilon_i: \epsilon_{max} \geq \epsilon_i$$

Ist der maximale Abstand zweier Soll-Punkte,  $\epsilon_{max}$ , der durch Rechenunterschiede herrührt, sehr viel kleiner als  $d_{max}$ , so kann man die Ungenauigkeiten der Soll-Position vernachlässigen. Genaue Werte, bei denen die Resultate noch akzeptabel sind, müssen aber empirisch für einen konkreten Fall ermittelt werden.

Bei einem Kartenbasierten Verfahren ist unbedingt zu verhindern, daß es zur Auswahl einer anderen Kartenkante beim Client als bei Server kommt, weil dies auf Grund des Matchings zur ausgewählten Kante gänzlich andere Soll-Positionen zur Folge hätte.

Wenn eine gänzliche Übereinstimmung zwischen den auf der Client- und der Serverseite berechneten Soll-Positionen gewünscht ist, ist es erforderlich, eine „Adaptive Arithmetik“ einzusetzen. Unter „Adaptiver Arithmetik“ ist gemeint, Module oder Klassen einzuführen, welche die Arithmetik eines bestimmten Prozessortyps nachbilden. Dabei reicht es, wenn nur eine der beiden Seiten die Arithmetik des Kommunikationspartners auf diese Art nachbildet. Da davon ausgegangen werden kann, daß ein Server im Vergleich zu einem Mobilen Objekt deutlich mehr Rechenleistung zur Verfügung hat, ist es sinnvoll, die Adaptive Arithmetik serverseitig zu implementieren.

Um jeweils die richtige Arithmetik-Klasse einzusetzen, muß das Mobile Objekt beim Aufsetzen einer Verbindung, dem Server den verwendeten Prozessortyp oder die

<sup>4</sup> Zur Erinnerung: Ist der Abstand zwischen Ist-Position und Soll-Position größer als  $d_{max}$ , verschickt der Client ein Update an den Lokationsserver.

Arithmetikklassse zur Korrektur mitteilen. Besitzt ein mobiles Objekt einen Prozessor vom Typ „XY“ teilt er es dem Server mit. Der Server benutzt darauf hin das Korrekturmodul für den Prozessor „XY“, das die Arithmetik des Prozessors in Software nachbildet. Dadurch wird erreicht, daß Client und Server gleiche Soll-Resultate berechnen.

Eine Alternative zur Adaptiven Arithmetik, die die Arithmetik einer bestimmten Hardware nachbildet, wäre es, generell sämtliche Berechnungen in selbstgeschriebenen einheitlichen Arithmetikklassen nachzubilden. Der Vorteil dieser Lösung wäre es, daß man nicht für jeden Prozessor, den man unterstützen möchte, eine spezielle Korrekturklasse benötigt. Diesem Vorteil stehen aber große Nachteile gegenüber. Die Berechnungen müßten dann sowohl vom Server als auch vom Client in Software erfolgen. Bei der Adaptiven Arithmetik dagegen genügt es, wenn der Server den Zusatzaufwand betreibt. Zusätzlicher Rechenaufwand für den Client bedeutet aber auch, daß der Client einen leistungsfähigeren Prozessor benötigt, der diesen bewältigen kann. Das hat wiederum zu Folge, daß der Stromverbrauch steigt, was bei akkubetriebenen Geräten die Laufzeit verringert! Darüber hinaus gilt generell, daß Software um Größenordnungen langsamer ist als Hardware. Der weitestgehende Verzicht des Einsatzes der prozessorigenen Arithmetik zur Implementierung hardwareunabhängiger Arithmetik wird durch den Nachteil eines Geschwindigkeitsnachteils erkaufte.

Bei der Programmiersprache Java, die übrigens auch für die Implementierung der Simulation eingesetzt wurde, verlangt die Spezifikation [JAVA], daß die Laufzeitumgebung (Virtual Machine) egal welcher Portierung IEEE-konforme Fließkomma-Formate benutzen muß. Allerdings wird diese Festlegung dadurch abgeschwächt, daß nicht nur die beiden oben erwähnten Formate für einfache und doppelte Genauigkeit eingesetzt werden dürfen, sondern auch sogenannte „extended“-Formate. Bei diesen sind die Fließkommazahlen mit noch mehr Bits codiert. Folglich ist auch bei einheitlicher Verwendung von Java beim Client und beim Server ggf. mit Abweichungen bei den Ergebnissen zu rechnen!

### 3.2.2 Grad der Übereinstimmung der Daten zwischen Client und Server

Im Kapitel 3.2.1 wurde der Einsatz verschiedener Prozessortypen auf Client- und Serverseite diskutiert. In diesem Kapitel geht es um die Frage, in wie weit beim kartenbasierten Verfahren die Kartendaten der beiden Kommunikationspartner übereinstimmen müssen und wie man geeignete Karten effizient aushandeln kann.

Mathematisch ausgedrückt müssen die Kartendaten auf dem Client eine Teilmenge der Kartendaten auf dem Server sein:  $Karte_{Client} \subseteq Karte_{Server}$ . Das gilt aber nur dann, wenn der Server genau weiß, welche Daten (Punkte und Kanten) die Karte des Clients enthält. Ist dieses Wissen nicht gegeben, müssen die beiden Karten identisch sein:  $Karte_{Client} = Karte_{Server}$

Doch was bedeutet das für den praktischen Einsatz? Eine gegebene Karte  $K$ , die der Server kennt, muß entweder als 1-zu-1-Kopie auf einem Client vorhanden sein oder die Kartendaten des Clients ergeben sich durch Weglassung von Kanten von der Originalkarte. Der letztere Fall ist etwas problematisch, weil der Server genau wissen muß, welche Kanten der Client kennt und welche nicht!

Um diese Kenntnis zu erlangen, muß im Verlauf der Kommunikation (sinnvoller Weise beim Verbindungsaufbau) zwischen Client und Server ausgehandelt werden, welche Daten beiden

bekannt sind. Da es vom Datenvolumen und der Übertragungszeit her vollkommen ineffizient wäre, eine ganze Karte zwischen den Kommunikationspartnern über einen Funkkanal zu übertragen oder Kante für Kante durchzugehen, um festzustellen, welche Kanten beide kennen oder nicht, muß ein anderer Weg gefunden werden. Dabei bietet sich an, einer Karte

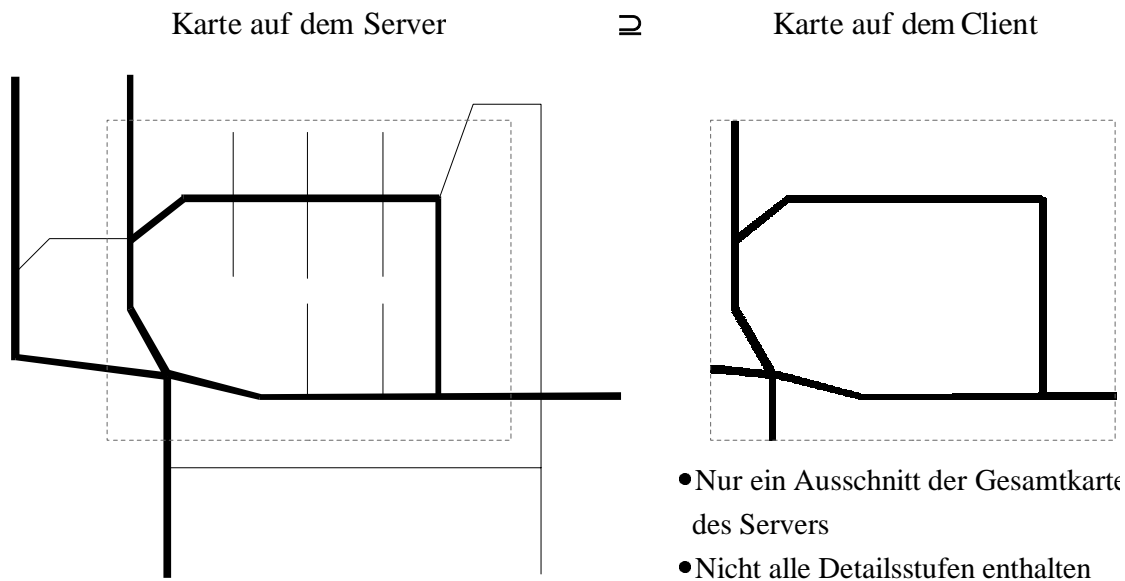


Abbildung 9 – Illustration der Sachverhalte der Kartendaten bei Client und Server. Der Client hat eine Karte, die einer Teilmenge der Karte auf dem Server entspricht. Die Karte des Clients wird charakterisiert durch den Ausschnitt (gestricheltes Rechteck) und der Detailstufe (unterschiedliche Strichdicken beim Server stehen für andere Hierarchiestufen). Im obigen Beispiel kennt der Client nur die höchste Ebene der Daten.

einen eindeutigen Bezeichner zu geben, der immer gleich bleibt und global eindeutig ist. Dieser Bezeichner könnte beispielsweise der Hash-Wert über die Gesamtdaten der Karte sein. Ist der Hash-Algorithmus geschickt gewählt, so kann man mit einer sehr hohen Wahrscheinlichkeit davon ausgehen, daß ein Bezeichner eindeutig ist. Ein solcher Hash-Algorithmus wäre z.B. MD-5 (siehe dazu [RIV92]).

Um aber zu realisieren, daß der Client nicht immer nur identische (vollständige) Karten wie ein Server besitzen muß (um Speicher auf der Clientseite zu sparen), wird noch ein Mechanismus benötigt, um effizient dem Server mitzuteilen, welche Daten bekannt sind und welche nicht. Hierzu wird ein kombiniertes Vorgehen vorgeschlagen, daß eine Mischung aus der Benennung des bekannten Ausschnittes und der Detailstufe ist. So können Mobile Geräte mit weniger Speicher eine passende Karte wählen. Der Ausschnitt kann im einfachsten Fall über die Koordinaten eines Rechtecks festgelegt werden, das den Bereich der Gesamtkarte kennzeichnet, der dem Client bekannt ist. Darüber hinaus ist es selbstverständlich möglich, ein Protokoll einzusetzen, das auch beliebige geschlossene Polygonzüge zuläßt. Die Detailstufe basiert auf einer einzuführenden Hierarchie auf den Daten. Dabei partitioniert man die Daten in verschiedene Klassen in einer hierarchischen Anordnung. Es könnte sich beispielsweise folgende Gliederung anbieten:

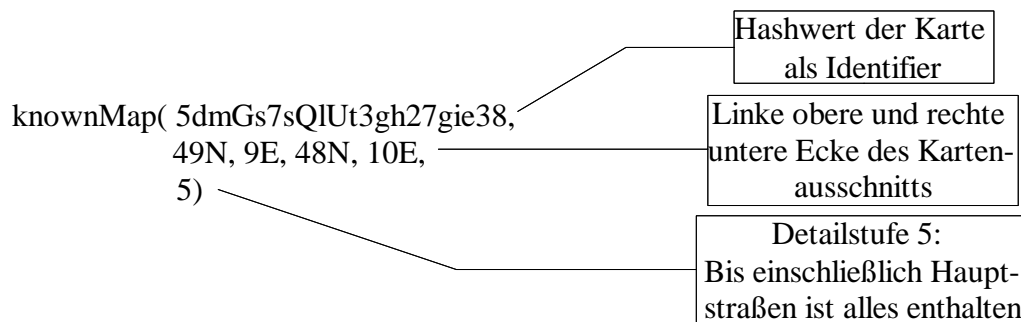
- |                 |                       |
|-----------------|-----------------------|
| 1. Autobahn     | 5. Hauptstraße        |
| 2. Bundesstraße | 6. Nebenstraße        |
| 3. Landesstraße | 7. kleine Nebenstraße |
| 4. Kreisstraße  |                       |

Diese Gliederung kann natürlich je nach Bedürfnis beliebig verfeinert oder verändert werden. Eine Detailstufe schließt dabei alle kleineren Stufen mit ein. Die Angabe einer Detailstufe von 3 bedeutet, daß die Karte Autobahnen, Bundesstraßen und Landesstraßen enthält, aber Kreisstraßen bereits ausgeschlossen sind.

Um dem Server mitzuteilen, welche Daten ein Client besitzt, muß er diesem also eine Nachricht schicken, die aus folgenden drei Komponenten besteht:

1. *Karten-ID*  
Eindeutiger Bezeichner einer Karte (beispielsweise Hash-Wert über Kartendaten)
2. *Bereich*  
Beschreibung des geographischen Bereiches, der von der Karte abgedeckt wird.
3. *Detailstufe*  
Zahl, welche die Detailstufe der Karte angibt

Ein Client könnte dem Server beispielsweise über folgende Nachricht mitteilen, welche Karte er kennt und benutzen will:



Auf diese Mitteilung hin muß der Server entweder diese Karte akzeptieren oder ablehnen, je nachdem, ob er sie in seiner Datenbank hat oder nicht.

Nach dem Akzeptieren einer Karte, bei der Anwendung des kartenbasierten Algorithmus, benutzt der Server nur den Teil der Gesamtkarte, welcher der Spezifikation durch den Client entspricht. Darüber hinausgehende Daten werden ignoriert. Bei dem obigen Beispiel würde das bedeuten, daß er keine Nebenstraßen heranziehen darf oder beim Verlassen des Kartenausschnittes kein Map-Matching mehr betreiben kann, sondern auf einen anderen Algorithmus übergehen muß.

### 3.2.3 Übertragung der Daten zum Server

Ein wichtiger Gesichtspunkt, der beachtet werden muß, ist die Datenübertragung. Es muß eine möglichst geeignete Kommunikationsmöglichkeit gefunden werden. Der Kommunikationskanal muß möglichst überall dort verfügbar sein, wo sich die Mobilien Objekte aufhalten können. Zudem sollte es möglich sein, die Update-Nachrichten möglichst zeitnah (also ohne langwierigen Verbindungsaufbau vor jeder Nachrichtenübermittlung) zu

übermitteln. Ein weiteres Kriterium ist natürlich der Preis, den man für die Kommunikation bezahlen muß.

In Deutschland und West-Europa sind die gut ausgebauten GSM-Mobilfunknetze ein nahezu überall zugängliches Medium zur Datenübertragung. Daher bieten sie sich in erster Linie an und werden hier näher diskutiert. Zur Auswahl stehen SMS, der Datendienst von GSM, HSCSD und GPRS. Daran anschließend wird noch auf Wireless-LAN gemäß IEEE 802.11 eingegangen.

Um Daten zu übertragen, ist der Datendienst von GSM gedacht. Man baut eine Verbindung wie mit einem gewöhnlichen Modem auf und kann dann Daten übertragen. Es handelt sich dabei um einen Dienst mit einer Bandbreite vom 9,6kBit/s bis 14,4kBit/s – je nach Netz. Die Bandbreite ist ausreichend für die relativ kleinen Update-Nachrichten. Vorteilhaft bei dieser Kommunikation ist, daß die Daten des Mobilobjektes garantiert unmittelbar nach der Übertragung beim Server ankommen (sofern die Kommunikation selbst erfolgreich war), weil die Daten nicht im Netzwerk zwischengespeichert werden, sondern eine Einwahlverbindung zum Netzwerk hergestellt wird. Dagegen sind die Anwahl und der Verbindungsaufbau der größte Nachteil dieses Verfahrens, weil etliche Sekunden vergehen bis die Verbindung steht. Ein Verbindungsaufbau dauert etwa 30 Sekunden! Das bedeutet, daß zwischen dem Erkennen, daß ein Lageupdate verschickt werden muß, und dem Zeitpunkt des Versendens sehr viel Zeit vergeht. Das hat zu Folge, daß sich während dieser Zeit die Soll-Positionen (durch den Algorithmus berechnet) von den Ist-Positionen (durch das Lagesystem bestimmt) mehr als die geforderte maximale Ungenauigkeit  $d_{max}$  unterscheiden.

Neben dem oben vorgestellten GSM-Datendienst gibt es seit ein paar Monaten noch HSCSD. Dabei handelt es sich im Prinzip um nichts anderes als den Datendienst, bei dem mehrere GSM-Kanäle gebündelt werden, um eine höhere Bandbreite zu erreichen. Aber auch bei HSCSD ist vor jeder Datenübertragung ein zeitraubender Verbindungsaufbau nötig.

Um dem langwierigen Verbindungsaufbau vor jeder Update-Nachricht zu entgehen, besteht natürlich die Möglichkeit, die Verbindung dauerhaft bestehen zu lassen. Da aber sowohl HSCSD als auch der normale GSM-Datendienst von allen Netz-Betreibern nur zeitabhängig abgerechnet werden, wäre das sehr teuer!

Ein Dienst, der nur nach übertragener Datenmenge abgerechnet wird und keinen zeitraubenden Verbindungsaufbau vorgeschaltet hat, ist SMS – der Short Message Service. Dabei werden Datenpakete zu je 160 Zeichen Nutzdaten übertragen. Wenn es möglich ist, alle Daten einer Update-Nachricht in eine SMS zu packen, kann dieser Dienst benutzt werden. Ansonsten muß man sich ein Protokoll überlegen, das es erlaubt, die Update-Nachrichten beim Client auf mehrere SMS zu verteilen und beim Server wieder zusammensetzen. Je nach Netzbetreiber kostet eine einzelne SMS bis zu 0,39DM, was sich bei der Übertragung von vielen Updates zu einem nicht unerheblichen Betrag aufsummiert.

Der gravierende Nachteil von SMS-Nachrichten ist aber, daß es keinerlei Garantien über die Laufzeiten gibt. Die eigene Erfahrung zeigt zwar, daß die Nachrichten meistens innerhalb von ein paar wenigen Sekunden beim Empfänger ankommen (auch von einem Netzbetreiber zum anderen), aber es kann auch vorkommen, daß SMS etliche Minuten bis zur Auslieferung unterwegs sind. Diese Varianz in den Laufzeiten macht den Dienst nur eingeschränkt für die Übertragung von Update-Nachrichten nutzbar.

Seit geraumer Zeit wird von den Netzbetreibern ein weiterer Dienst namens GPRS angeboten. Dabei handelt es sich um einen paketorientierten Dienst. Abgerechnet wird nach Datenvolumen (je nach Netzbetreiber kommt aber noch eine geringe zeitabhängige Tarif-Komponente hinzu). Die Bandbreite beträgt bis zu 53kBit/s, soll aber in Zukunft auch bis zu 107kBit/s erlauben (Quelle: [D2MAN]). Es ist möglich, einmal aufgebaute Verbindungen zu parken, so daß kein erneuter Verbindungsaufbau nötig ist, um erneut Daten zu übertragen. GPRS ist daher als Übertragungsmedium am besten geeignet, da es sowohl eine unmittelbare Datenübertragung (also keine unvorhersehbaren Datenlaufzeiten durch Zwischenspeicherungen im Netz wie bei SMS) als auch ein kostengünstiges dauerhaftes Online-Bleiben wegen der Volumenabrechnung erlaubt, um den zeitraubenden Verbindungsaufbau zu sparen.

Durch GPRS ist es ohne Aufbau eines eigenen Kommunikationsnetzes möglich, an fast jedem beliebigen Ort heute schon Lokationsdienste anzubieten, die bezahlbar bleiben.

Neben den bisher erwähnten Mobilfunknetzen soll noch eine andere Netzgattung erwähnt werden: Wireless-LAN nach IEEE 802.11. Funk-LANs werden zunehmend in der Praxis eingesetzt, um sich die Verkabelung eines konventionellen LANs zu sparen. Daher sind diese auch immer häufiger vorzufinden. So liegt es nahe, diese drahtlosen Netzwerke für die Umsetzung von ortsbezogenen Diensten einzusetzen. Im Gegensatz zu den Mobilfunknetzen sind die Wireless-LANs lokal begrenzt. Meist steht das LAN nur innerhalb eines Gebäudes oder in dessen unmittelbaren Umgebung zur Verfügung. Dadurch wird entsprechend der Einsatzradius der Mobilobjekte beschränkt. Das bedeutet, daß Wireless-LAN nur für Indoor-Anwendungen eingesetzt werden kann.

Aber auch wenn man im (eigenen) LAN nicht nach Zeit oder Datenvolumen tarifiert wird wie bei den Mobilfunknetzen, ist es sinnvoll, das Datenaufkommen eines Teilnehmers möglichst gering zu halten, um den anderen Teilnehmern noch für ihre Zwecke Netzkapazitäten frei zu lassen. In einem anderen Szenario, bei dem das Funk-LAN ausschließlich für ortsbezogene Dienste zur Verfügung steht, ist es durch den Einsatz der hier vorgestellten Koppelnavigations-Verfahren möglich mehr Mobile Objekte mit dem Server kommunizieren zu lassen (bei gleicher Bandbreite des LAN).

Bei gleicher belegter Bandbreite könnte man auch eine höhere Genauigkeit der Positionsinformationen (also  $d_{max}$  verkleinern) benutzen. Durch Verringerung des Toleranzradius steigt nämlich die absolute Zahl an Updates an. Erreicht man die gleiche Zahl wie ohne Dead-Reckoning, hat man eine bessere Auflösung bei gleichem Datenumfang.

Folglich ist es auch bei der Verwendung eines Funk-LANs vorteilhaft, den Zusatzaufwand für die Implementierung eines Dead-Reckoning-Protokolls auf sich zu nehmen.

### 3.2.4 Erkennung von Funkschatten oder Verbindungsabbrüchen

Ein immer wieder auftretendes Problem bei Mobilobjekten in Verbindung mit drahtloser Kommunikation sind Verbindungsabbrüche. Ursachen dafür sind beispielsweise die Einfahrt eines Fahrzeugs in einen Tunnel oder das Betreten eines Hauses, in dem das Funksignal durch die Wände zu stark gedämpft wird. Auch hohe Gebäude können einen Funkschatten bilden, so daß es zum Abbruch der Verbindung kommt. In jedem Fall muß das Protokoll in der Lage sein, mit Verbindungsabbrüchen zurechtzukommen. Das bedeutet vorallem, Abbrüche zu

erkennen.

Wenn der Server einen Verbindungsabbruch erkennt, kann er bei einer Anfrage nach der Position des betreffenden Objektes entweder eine Fehlermeldung liefern oder die letzte bekannte Position (aus dem letzten Lageupdate) ausgeben. Allerdings muß dabei dem Anfragenden kenntlich gemacht werden, daß diese Position nicht aktuell ist.

Bleibt eine Updatenachricht wegen eines Verbindungsabbruches aus, geht der Server lediglich weiter davon aus, daß die Soll-Position um nicht mehr als die maximal zulässige Abweichung von der Ist-Position entfernt ist. Dem Server bleibt nichts anderes übrig als den Koppelnavigations-Algorithmus fortzusetzen und weitere Soll-Positionen zu berechnen und ggf. an anfragende Clients zu melden. Das Problem ist, daß der Server immer passiv auf Nachrichten wartet und daher nicht erkennen kann, wann zwar versucht wurde, eine Nachricht zu senden, aber die Kommunikation nicht möglich war. Um trotzdem Verbindungsabbrüche erkennen zu können, ist es erforderlich das Protokoll leicht zu verändern.

Eine Möglichkeit ist es, einen Timeout  $T_{send}$  einzuführen. Das Mobile Objekt wird nun verpflichtet spätestens nach Ablauf von  $T_{send}$  eine Nachricht an den Server zu schicken. Diese Nachricht muß gesendet werden, auch wenn die Soll-Positionen noch innerhalb des Toleranzbereiches von den Ist-Positionen sind. Verstreicht die Zeit  $T_{send}$ , ohne daß der Server eine Nachricht empfängt, bedeutet dies, daß die Verbindung zum Client abgebrochen ist. Als Nachricht kann man entweder eine komplette Updatenachricht verschicken, was gleichzeitig auch die Daten auf dem Server auf den aktuellsten Stand bringt, oder eine Ping-Nachricht. Unter einer Ping-Nachricht ist eine Nachricht zu verstehen, die vom Datenvolumen sehr klein ist und keine Informationen übermittelt. Sie dient lediglich dazu, dem Server zu zeigen, daß die Verbindung zwischen Client und Server noch besteht. Je nach zur Verfügung stehendem Kommunikationsmedium muß bestimmt werden, ob es vorteilhafter ist, eine Updatenachricht oder eine Ping-Nachricht zu verschicken. Meist ist allerdings der Overhead für das Senden einer Nachricht größer als das Datenvolumen, so daß der Inhalt keine Rolle mehr spielt.

In [WOL99] wird ein anderer Weg vorgeschlagen: Mit jeder neu berechneten Soll-Position wird der Radius des Toleranzbereiches verkleinert. Das hat zu Folge, daß nach einer endlichen (und vorher berechenbaren) Anzahl von Schritten, das Toleranzband so klein geworden ist, daß eine Updatenachricht verschickt werden muß. Bleibt die Updatenachricht aus, erfährt so der Server von einem Abbruch der Verbindung. Im Prinzip realisiert dieser Ansatz auch einen Timeout. Der Unterschied ist nur, daß die Timeout-Zeit implizit durch die Anzahl der benötigten Schritte bis zur Reduktion des Toleranzradius auf 0 vorgegeben wird und nicht explizit als Zeitwert vorgegeben ist. Man kann aber aus der Anzahl der Soll-Positionsberechnungen und der Rate mit der eine Neuberechnung erfolgt, auch für dieses Verfahren eine Timeout-Zeit errechnen.

Als letztes Verfahren soll ein Algorithmus vorgestellt werden, der nur beim kartenbasierten Ansatz funktioniert. Annahme: ein Mobiles Objekt bewegt sich mit einer Geschwindigkeit ungleich 0 längs einer Kartenkante. Solange sich das Mobile Objekt annähernd gleichförmig bewegt, besteht kein Bedürfnis, eine Updatenachricht zu verschicken. Verändert sich nun aber das Verhalten des Objektes – beispielsweise weil es vor einer Ampel halten muß, ist es nötig, ein Update zu verschicken. Bricht nun die Verbindung ab, hat der Server wie in den anderen Fällen keine Möglichkeit, dies zu erkennen. An dieser Stelle kann die Karte hilfreich sein. Da

er keine Updatenachricht empfangen hat, geht der Server davon aus, daß er weiterhin unverändert Soll-Positionen anhand der ihm bekannten Daten berechnen kann. Da die Geschwindigkeit ungleich 0 ist, verlassen irgendwann die Soll-Positionen die momentane Kante der Karte. Befinden sich schließlich die Soll-Positionen so weit außerhalb der Kante, daß sie außerhalb des Toleranzradius  $d_{max}$  sind, kann der Server von einem Verbindungsabbruch ausgehen, da er nach dem Verlassen der Kante eine Updatenachricht hätte empfangen müssen.

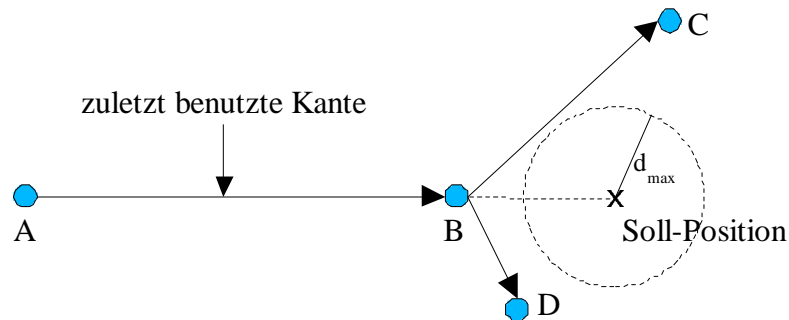


Abbildung 10 – Erkennung des Verbindungsabbruchs, weil die Soll-Position bei Fortführung des Verfahrens zu einer Position führt, die mehr als  $d_{max}$  vom Kantenpunkt B ist. Wenn keine Updatenachricht ankommt, die anzeigt, welche der beiden Ausgangskanten gewählt wurde, liegt ein Verbindungsabbruch vor.

Problematisch ist allerdings, wenn die Kante nur eine einzige Nachfolgekante besitzt, wenn Routenplanung eingesetzt wird oder wenn durch irgendeine andere Regel beim Fehlen einer neuen Updatenachricht automatisch eine bestimmte Ausgangskante gewählt wird. In diesem Fall würde diese hier beschriebene Erkennungsmethode versagen, weil lediglich angenommen würde, daß sich das Mobile Objekt genau so weiterverhält wie bisher prognostiziert.

### 3.2.5 Grad der Uhrensynchronität

Der Client verschickt in seiner Updatenachricht an den Server einen Zeitstempel  $T_{Update}$ , durch welchen er dem Server mitteilt, zu welchem Zeitpunkt die Daten des Updates berechnet wurden. Dieser Zeitstempel wird vom Server benötigt, um die Soll-Positionen zu berechnen. Vereinfacht ausgedrückt muß zur Berechnung der momentanen Soll-Position die zurückgelegte Strecke  $\Delta s$  ermittelt werden, die ausgehend von der letzten Position vom Mobilien Objekt zurückgelegt wurde.

Die Strecke wird nach einer einfachen Formel berechnet:  $\Delta s = v \cdot \Delta t$  (1)

Dabei ist die Geschwindigkeit  $v$  aus dem letzten Update bekannt, da sie explizit in der Updatenachricht enthalten ist (siehe Kapitel 4.1). Die verstrichene Zeit  $\Delta t$  muß dagegen berechnet werden:  $\Delta t = T_{aktuell} - T_{Update}$  (2)

$T_{aktuell}$  ist die aktuelle Zeit, die der Server von seiner internen Uhr ablesen kann.  $T_{Update}$  ist die Zeit, zu der die Updatenachricht vom Client erstellt wurde. Die Zeit hat der Client aus von seiner Uhr abgelesen.

Darin liegt ein nicht unerhebliches Problem, weil mit absoluten Zeiten gerechnet wird, die von verschiedenen Uhren abgelesen wurden!<sup>5</sup>

Ideal wäre es natürlich, wenn beide Uhren synchron laufen würden. Das ist aber eine Forderung, die in der Praxis – vor allem bei sehr vielen zu synchronisierenden Uhren – ausgeschlossen ist. Die Uhren zwischen dem Server und allen angeschlossenen Clients müßten stets synchron laufen. Bei sehr vielen Mobilien Objekten wäre das ein riesiger Kommunikationsaufwand, um die Uhren zwischen allen Beteiligten zu synchronisieren.

Allerdings reicht auch eine viel schwächere Forderung für die korrekte Funktion der Dead-Reckoning-Verfahren. Es muß lediglich gelten, daß die Uhrenrate des Servers und die Uhrenrate des Clients möglichst geringen Drift aufweisen. Diese Forderung ist von jeder heute eingesetzten Quarzuhr erfüllt (Driftrate gegenüber einer idealen Uhr ist  $10^{-9}$ ). Ein gewisser Drift ist zu tolerieren, weil immer wieder neue Updates geschickt werden, so daß der örtliche Abweichung bei der Berechnung der Soll-Positionen immer wieder auf Null gesetzt wird. Erst wenn lange Zeit kein neues Update ankommt, macht sich die örtliche Abweichung immer stärker wegen der zunehmenden Abweichung der Uhren zwischen Client und Server bemerkbar (Wegen  $s=vt$ ).

Noch zu lösen ist aber das Problem der absoluten Abweichung (Offset) zwischen den Uhren. Dazu ist die Formel (2) zu modifizieren zu:  $\Delta t = (T_{\text{aktuell}} - T_{\text{Ankunft}}) + T_{\text{Komm}}$  (3)

Statt  $T_{\text{Update}}$  wird nun die Ankunftszeit der Updatenachricht beim Server  $T_{\text{Ankunft}}$  benutzt. Durch Subtraktion von  $T_{\text{Ankunft}}$  von  $T_{\text{aktuell}}$  erhält man so die verstrichene Zeit, seit der die Updatenachricht beim Server angekommen ist. Da beide Zeiten von der gleichen Uhr abgelesen wurden, gibt es keine Probleme mit der Uhrensynchronität.

Um die tatsächlich verstrichene Zeit zu erhalten, seit dem der Client die Updatenachricht erstellt hat, muß man dazu die Kommunikationszeit  $T_{\text{Komm}}$  addieren.  $T_{\text{Komm}}$  ist dabei die Dauer, die eine Nachricht vom Zeitpunkt des Abschickens beim Client zur Ankunft beim Server über den Kommunikationskanal benötigt. Zur Bestimmung dieser Zeit, kann der Server einmalig oder regelmäßig Ping-Nachrichten an den Client schicken. Aus der Zeitdauer bis eine Antwort auf die Ping-Nachricht  $T_{\text{Ping}}$  beim Server wieder ankommt, kann  $T_{\text{Komm}}$  berechnet werden.

$$\text{Es gilt: } T_{\text{Ping}} = 2 \cdot T_{\text{Komm}} + \epsilon \text{ somit } T_{\text{Komm}} = \frac{T_{\text{Ping}} - \epsilon}{2}$$

Dabei ist  $\epsilon$  die Bearbeitungszeit, die der Client benötigt, um auf die Ping-Nachricht zu antworten. Ist diese Bearbeitungszeit gegenüber  $T_{\text{Komm}}$  sehr klein, kann sie auch vernachlässigt werden.

Ist  $T_{\text{Komm}}$  gegenüber  $T_{\text{aktuell}} - T_{\text{Ankunft}}$  sehr klein, kann  $T_{\text{Komm}}$  ebenfalls vernachlässigt werden und die vereinfachte Formel für  $\Delta t$  kann eingesetzt werden:  $\Delta t = (T_{\text{aktuell}} - T_{\text{Ankunft}})$  (4)

Durch den Einsatz der Formel (3) statt (2), kann man auf aufwendige Protokolle zur Uhrensynchronisation verzichten. Wenn es das Netzwerk auf Grund der kurzen Kommunikationszeiten erlaubt, ist es sogar möglich, auf die einfachere Formel (4)

<sup>5</sup> Auch der Client muß Soll-Positionen berechnen (um eine Abweichung von der Ist-Position feststellen zu können). Das erfolgt nach dem gleichen Schema wie beim Server. Aber dabei kommt jedes Mal die eigene Uhr zum Einsatz, so daß es keine Synchronisationsprobleme gibt!

zurückzugreifen und sich dadurch auch den Aufwand zu sparen, die Antwortzeiten auszumessen.

### 3.3 Ausblick

Die bisher vorgestellten Verfahren sollen eine Übersicht über die Möglichkeiten für den Einsatz von Koppelnavigation in einem Lokationsdienst zur Einsparung von Update-Nachrichten im Vergleich zu dem Fall geben, bei dem keine Koppelnavigation eingesetzt wird. Im weiteren Verlauf der Diplomarbeit werden zwei Verfahren näher betrachtet. Das ist zum einen der einfachste Fall: die lineare Fortsetzung. Dieses Verfahren liefert einen Vergleichspunkt für alle anderen Verfahren, die eine Verbesserung der Resultate gegenüber dem linearen Fortsetzen erzielen wollen. Außerdem kann dieses Protokoll als Rückfallalgorithmus dienen, weil es immer benutzt werden kann. Zum anderen wird das kartenbasierte Verfahren untersucht. Es soll insbesondere verdeutlicht werden wie groß die Verbesserung zum linearen Fortsetzen ist und mit welchem zusätzlichen Aufwand diese Verbesserung verbunden ist.

Die anderen Verfahren werden nicht weiter untersucht, da zu erwarten ist, daß sie von den erzielbaren Resultaten zwischen dem linearen Fortsetzen und dem kartenbasierten Verfahren liegen. Die Krümmungsapproximation ist eigentlich eine Erweiterung der linearen Fortsetzung um Kurven zur Optimierung der zukünftigen Streckenmodellierung. Das bewirkt, daß nicht nur geradlinige Bewegungen vom Koppelnavigationsprotokoll abgedeckt werden, sondern auch Kurvenfahrten besser (also mit weniger Updates) verfolgt werden können. Das historienbasierte Verfahren ist, nach der Aufstellung einer Karte anhand der gegebenen Historien, ein hybrides Verfahren, das eine Mischung zwischen einem Fortsetzungsverfahren und einem Kartenverfahren darstellt. Aus den Historien wird nämlich eine Karte gewonnen, die allerdings im Gegensatz zum Kartenansatz nur eine bevorzugte Bewegungsrichtung modelliert, nicht aber ausschließliche Wege aufzeigt<sup>6</sup>. Wird das Historienbasierte Verfahren streng ausgelegt, so daß angenommen wird, daß die Bewegung der Objekte stets entlang der aus den Historien gewonnenen Karte erfolgt, sollte es die gleichen Ergebnisse erzielen wie der kartenbasierte Ansatz.

Die Kartenverfahren mit Wahrscheinlichkeiten und Routenplanung sollten geringfügig bessere Ergebnisse erzielen als der einfache Kartenalgorithmus. Sie decken zwar den Bewegungspfad besser ab als das einfache Verfahren, aber sie können nicht den Geschwindigkeitsverlauf vorhersagen. Viele Updates stammen aber daher, daß sich das Objekt nicht mit konstanter Geschwindigkeit bewegt, sondern eine Schwankung auftritt. (Z.B. ein Anhalten eines Fahrzeugs, um anderen Autos Vorfahrt zu gewähren) Daher sollten die Ergebnissen aller Kartenverfahren sehr ähnlich sein.

---

<sup>6</sup> Bei einem Kartenverfahren wird primär davon ausgegangen, daß sich ein Mobiles Objekt nur entlang der durch die Karte vorgegebenen Kanten bewegen kann. Nur in Sonderfällen, beispielsweise wenn der Kartenbereich verlassen wird, muß ein anderes Verfahren eingesetzt werden! (Diese Sonderfallbehandlung wird aber später noch erläutert!)

## 4 Gemeinsame Grundbestandteile aller Verfahren

---

In diesem Kapitel werden alle Elemente beschrieben, die alle Dead-Reckoning-Verfahren gemeinsam haben. Dabei geht es sowohl um Algorithmen und Methoden wie beispielsweise den Endlosschleifen, in denen Client und Server immer wieder ihre Daten berechnen, als auch um die Festlegung von Datenstrukturen, die benötigt werden.

### 4.1 Struktur von Updatenachrichten

Eine Updatenachricht dient dazu, dem Lokationsdienst aktuelle Informationen über die Position des mobilen Objektes zukommen zu lassen. Sie sind das Bindeglied zwischen Client und Server. Eine Updatenachricht muß immer dann gesendet werden, wenn die anhand des Dead-Reckoning-Verfahrens berechnete Soll-Position von der tatsächlichen Ist-Position um mehr als einen festgelegten Schwellwert  $d_{max}$  abweicht. Da sowohl der Client als auch der Server immer die gleichen Daten haben müssen damit der Koppelnavigationsalgorithmus auf beiden Seiten die gleichen Ergebnisse liefert, dürfen einzelne Updates weder verloren gehen noch doppelt ankommen. Das Übertragungsprotokoll des Kommunikationskanals muß folglich eine verlustfreie Kommunikation ohne Nachrichtenverdopplung gewährleisten!

Unabhängig vom verwendeten Koppelnavigationsalgorithmus besteht eine solche Updatenachricht aus folgenden Bestandteilen:

1. *Objekt-ID*  
Eindeutiger Bezeichner, der das sendende Objekt identifiziert.  
(Im allgemeinen kann davon ausgegangen werden, daß ein Lokationsserver mehr als ein Objekt verfolgt, so daß er beim Empfang einer Updatenachricht wissen muß, welcher Datenbestand aktualisiert werden muß.)
2. *Zeitpunkt T*  
Hierbei handelt es sich um den Zeitpunkt, an dem die Daten – d.h. die Position des Objektes – bestimmt wurden. Diese Angabe ist nötig, weil das Senden der Nachricht durch das mobile Objekt bzw. die Verarbeitung der Nachricht im Lokationsdienst nicht in Nullzeit erfolgen kann. Dieser Zeitpunkt ist dann der Ausgangspunkt für die Berechnungen der Soll-Positionen.
3. *aktuelle Position des Objektes*  
Koordinaten, an denen sich das Objekt zum Zeitpunkt T befunden hat.  
Unterstütztes Koordinatensystem ist WGS84 (da von GPS verwendet).
4. *Richtungswinkel  $\alpha$*   
Bewegungsrichtung des mobilen Objektes in Grad. Dabei soll gelten:  $\alpha \in [0;360[$

0° entspricht einer Bewegung nach Norden. Der Richtungswinkel wird gegen den Uhrzeigersinn gezählt.

#### 5. *Geschwindigkeit v*

Betrag der Geschwindigkeit in m/s.

Je nach Verfahren kann man hier eine Momentan-, eine Maximal- oder eine Durchschnittsgeschwindigkeit angeben. D.h. die Angabe einer Geschwindigkeit ist unabhängig vom Verfahren notwendig, aber die Interpretation einer Angabe ist unterschiedlich!

In Pseudonotation hat eine Update-Nachricht folgende Struktur:

```
struct PosUpdate {
    objectID  OID,
    time      timestamp,
    Position  Pos,
    double    angle,
    double    v
}
```

## 4.2 Abstrakte Klasse *DRAAlgorithm*

In den Pseudocode-Darstellungen, die in diesem Kapitel noch folgen werden, müssen immer wieder Bezüge zum Dead-Reckoning-Algorithmus gemacht werden. Daher wird an dieser Stelle zuerst eine abstrakte Basisklasse „*DRAAlgorithm*“ vorgestellt, auf die Bezug genommen werden kann.

```
abstract class DRAAlgorithm {
    // fields
    objectID  OID;

    // public methods
    Position  correctCurrPos( Position );
    Position  computeCurrPos();
    boolean   deviationExceeded();
    PosUpdate computeNewUpdate();
    void      setNewUpdate( PosUpdate );
}
```

Zu „*DRAAlgorithm*“ ist ausdrücklich anzumerken, daß es sich hierbei um eine abstrakte Grundklasse handelt. Konkrete Implementierungen sind folglich abgeleitete Klassen. Eine derartige Subklasse kapselt (und implementiert) dann jeweils einen speziellen Dead-Reckoning-Algorithmus. Die Subklasse enthält neben der Programmlogik zur Berechnung der Soll-Position auch ggf. die aktuellen Zustandsdaten für ein jeweiliges Mobiles Objekt, die benötigt werden. Je nach Art des Verfahrens werden naturgemäß unterschiedliche Zusatzdaten benötigt. Während beispielsweise ein Verfahren, daß die bisherige Bewegung linear fortsetzt, nur Informationen zur Bewegungsrichtung und -geschwindigkeit benötigt (beide sind dem letzten Update zu entnehmen), wird bei einem kartenbasierten Verfahren zusätzlich noch eine Karte in elektronischer Form benötigt.

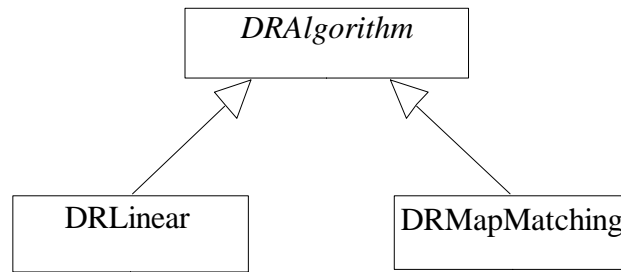


Abbildung 11- Beispiel für eine Vererbungshierarchie der Algorithmen in UML-Notation. DRAlgorithm ist die abstrakte Grundklasse. Die beiden abgeleiteten Klassen entsprechen konkreten Implementierungen zweier Dead-Reckoning-Algorithmen.

Das Attribut `OID` enthält den eindeutigen Bezeichner für die Algorithmus-Instanz. Anhand dieses Bezeichners wird ein Objekt identifiziert. Er wird für die Verwaltung der Instanzen auf dem Server benötigt. Der Algorithmus muß für jedes Mobile Objekt ablaufen. Dazu wird jeweils eine eigene Instanz benötigt, welche die Zustandsdaten speichert.

Mit der Methode „`computeCurrPos`“ wird der Algorithmus veranlaßt, die Soll-Position zu berechnen. Das erfolgt anhand der in der letzten Update-Nachricht enthaltenen Daten, der aktuellen Uhrzeit und ggf. unter Hinzunahme weiterer Daten (z.B. Kartendaten). Das Ergebnis an den Aufrufer zurückgegeben. Die Methode wird sowohl vom Client als auch vom Server aufgerufen.

Die Methode „`correctCurrPos`“ dient dazu, die im Parameter übergebene Position anhand des verwendeten Algorithmus zu korrigieren. Sie wird vom Client benötigt und wird nach der Bestimmung der Ist-Position aufgerufen. Eine derartige Korrektur wird beispielsweise beim kartenbasierten Verfahren benötigt. Dort wird angenommen, daß sich das Mobile Objekt immer nur auf Straßen aufhalten kann. Durch Ungenauigkeiten der Lagemessung (z.B. GPS) kann es vorkommen, daß die gemessene Lage neben einer Straße liegt. Bei der Korrektur würde dann die passende Straße gesucht und die Lage entsprechend so versetzt, daß sie sich nach der Korrektur auf der Straße befindet. Wird beim jeweiligen Verfahren keine derartige Korrektur benötigt, ist der Rückgabewert mit dem übergebenen Parameter identisch.

Anhand von „`deviationExceeded`“ wird das Objekt vom Client angestoßen, zu prüfen, ob die Soll-Position von der Ist-Position um mehr als die maximale Toleranzschwelle abweicht. Ein Rückgabewert von „`true`“ bedeutet, daß die Schwelle überschritten ist. Entsprechend steht „`false`“ dafür, daß es nicht nötig ist, eine neue Update-Nachricht zu verschicken.

Die Methode, „`computeNewUpdate`“, dient dazu, eine neue Update-Nachricht nach dem entsprechenden Verfahren berechnen zu lassen. Die Methode ruft der Client dann auf, wenn `deviationExceeded` den Wert `true` ergeben hat.

Die letzte Methode, „`setNewUpdate`“, wird nur vom Server benötigt. Sie teilt der Algorithmus-Instanz mit eine neue Updatenachricht mit. Das ist nötig, weil dem Server keine Ist-Positionen bekannt sind. Folglich kann er auch nicht selbst Updatenachrichten berechnen. Der Server bekommt vielmehr seine Updates vom Client. Da Updates aber für die

Berechnung der Soll-Position vom Algorithmus benötigt werden, muß es diese Methode geben.

Zwischen einem Mobilen Objekt und einer Algorithmus-Instanz besteht übrigens eine 1-zu-1-Beziehung.

Abschließend soll eine tabellarische Übersicht über die Methoden der Klasse *DRAlgorithm* gegeben werden, die verdeutlicht, welche Methoden vom Client bzw. vom Server benutzt werden:

<i> Methode </i>	<i> Client </i>	<i> Server </i>
correctCurrPos	x	
computeCurrPos	x	x
deviationExceeded	x	
computeNewUpdate	x	
setNewUpdate		x

### 4.3 Bestimmung der neuen Soll-Position

Zur Bestimmung der neuen Soll-Position werden unabhängig vom zugrundeliegenden Koppelnavigationsprotokoll folgende Informationen benötigt:

1. *aktuelle Zeit*  $T_{\text{aktuell}}$   
Nur durch die Kenntnis des Parameters der aktuellen Zeit ist es möglich, anhand des Dead-Reckoning-Algorithmus die Soll-Position zu berechnen.
2. Koordinaten der letzten *Soll-Position*  $P_{\text{alt}}$  (und weitere Daten wie Richtungsvektor, Momentangeschwindigkeit, etc.)
3. *Zeitpunkt* der letzten Soll-Position  $T_{\text{alt}}$

Das jeweilige Verfahren kann ausgehend von der letzten berechneten Position  $P_{\text{alt}}$  die neue Soll-Position berechnen, indem es die verstrichene Zeit  $\Delta t$  zwischen den beiden Zeitpunkten,  $T_{\text{aktuell}}$  und  $T_{\text{alt}}$ , berechnet und anhand der zur Verfügung stehenden Informationen (Kartendaten, Bewegungsrichtung, Geschwindigkeit, ...) die aktuelle Soll-Position bestimmt. Hierbei ergeben sich selbstverständlich Unterschiede zwischen den einzelnen Verfahren. Beispielsweise haben nur kartenbasierte Algorithmen Kartendaten. Andererseits benötigt jedes Verfahren Informationen zur Geschwindigkeit und zur Bewegungsrichtung. Diese stammen aus der letzten Updatenachricht, die sowohl der Client als auch der Server kennen.

Um eine Aufsummierung von Rundungsfehlern zu vermeiden, ist es erstrebenswert jede Soll-Positionsbestimmung ausgehend von der zuletzt an den Lokationsserver gesandten Ist-Position zu berechnen (also der Update-Nachricht). D.h. bei jeder Berechnung wird die Soll-Position anhand der Zeitdifferenz zwischen dem Zeitpunkt des letzten Sendens der Ist-Position an den Server und der aktuellen Zeit ermittelt.

Hierbei ist das Problem zu beachten, daß Client- und Serveruhr nicht synchron laufen. Eine

Diskussion zum Thema Uhrensynchronität ist im Kapitel 3.2.5 auf Seite 29 nachzulesen.

## 4.4 Mobiles Objekt (Client)

Das Mobile Objekt muß in einer Schleife immer wieder zuerst die aktuelle Lage (Ist-Position) anhand des Lagemeßsystems bestimmen und dann durch den jeweiligen Dead-Reckoning-Algorithmus die Soll-Lage berechnen. Ist beides geschehen, muß überprüft werden, ob die Ist-Lage von der Soll-Lage um mehr als die festgelegte maximale Toleranzschwelle  $d_{max}$  abweicht. Bei einer Überschreitung der Schwelle, muß eine Update-Nachricht an den Lokationsserver geschickt werden, um diesen von der neuen Position in Kenntnis zu setzen. Ist  $d_{max}$  noch nicht überschritten, muß lediglich gewartet werden bis das Lagesystem die Ist-Position erneut bestimmt hat. Dann beginnt die Schleife erneut.

Bei Verwendung von GPS ist das Intervall, nach welchem eine neue Ist-Position bestimmt wird, systembedingt bei 1 Sekunde festgelegt. Bei anderen Systemen muß man die jeweilige Spezifikation des Systems beachten.

Als Algorithmus gefaßt sieht die Schleife folgendermaßen aus:

```

loop
  P1 = determineCurrentPosition();
  P1 = myAlgorithm.correctCurrPos( P1 );           (*)
  P2 = myAlgorithm.computeCurrPos();
  if (myAlgorithm.deviationExceeded())
    then {
      newUpd = myAlgoritm.computeNewUpdate();
      send newUpd to location-server;
    } else {
      wait till location sensor has got a new position
    }
end loop

```

Nach der Bestimmung der Ist-Position wird in der Zeile (\*) der jeweilige Korrekturalgorithmus des benutzten Dead-Reckoning-Verfahrens angewandt, um beispielsweise beim Kartenbasierten Verfahren die Positionsmeldung des Meßsystems auf die Karte zu matchen. Wird beim jeweiligen Verfahren keine derartige Korrektur benötigt, verändert die Methode „correctCurrPos“  $PI$  nicht. (Also:  $PI'=PI$ )

Unterschiede in den Protokollen ergeben sich durch die Wahl des Algorithmus und durch die Wahl einer geeigneten Toleranzschwelle.

Durch Ausnutzung von Vererbung, Dynamischen Binden und Polymorphismus von objektorientierten Programmiersprachen ist es möglich, den selben Programmcode – wie er oben in Pseudonotation dargestellt ist – unabhängig vom Dead-Reckoning-Algorithmus einzusetzen.

## 4.5 Lokationsdienst (Server)

Der Server muß zwei parallele Tätigkeiten durchführen. Zum einen muß er Anfragen beantworten, die sich auf den momentanen Aufenthaltsort der überwachten Mobilien Objekte

beziehen. Zum anderen muß er auf Updatenachrichten seitens der Mobilien Objekte warten und diese ggf. verarbeiten.

Da diese beiden Aufgaben zeitgleich auf die Daten eines Mobilien Objektes zugreifen können müssen, sind bei einer Implementierung Sperrmechanismen vorzusehen, um die Konsistenz der Daten zu gewährleisten.

### 4.5.1 Datenbestand

Nachfolgend soll die Frage beantwortet werden: Welche Daten müssen gespeichert werden? Diese sind folgende:

- *Letzte Updatenachricht* des Mobilien Objektes  
(Der Inhalt der Updatenachricht ist im Kapitel 4.1 erläutert.)
- aktuell benutztes *Dead-Reckoning-Verfahren*  
(schließlich muß man zur Beantwortung der Lokationsanfragen die Soll-Position bestimmen. Das ist nur möglich, wenn das „richtige“ DR-Verfahren benutzt wird!)
- *Zustandsinformationen* zum Dead-Reckoning-Verfahren  
Je nach DR-Verfahren müssen zusätzliche Zustandsdaten gespeichert werden!  
Beim Kartenbasierten-Dead-Reckoning beispielsweise: Kartendaten, letzte gemappte Kante, letzter besuchter Knoten, Fallback-Algorithmus, wenn der Kartenbereich verlassen wird, etc.

Natürlich werden all diese Daten nicht nur beim Server, sondern auch beim Mobilien Objekt benötigt. Für die Funktion des Algorithmus muß auch das Mobile Objekt zwangsläufig sämtliche Daten kennen. Aber im Unterschied zum Client hat der Server die Daten von mehr als einem Mobilien Objekt zu verwalten. Dadurch ergeben sich andere Anforderungen an die Datenhaltung.

Der Zugriff auf die Daten des Servers muß konkurrent möglich sein. Daher ist sind Sperrmechanismen (Locks) nötig! Dieses Kriterium wird beispielsweise von einer Relationalen Datenbank erfüllt. Allerdings veralten die Daten durch die immer wiederkehrenden Updates schnell und müssen (zum Teil) einen Servercrash nicht überdauern. Das impliziert, daß persistente Datenhaltung für einen Teil der Daten nicht zwingend erforderlich ist!

Sei im folgenden „data“ der Name der Datenbanktabelle, in der die Instanzen von „*DRAAlgorithm*“ gespeichert werden. An dieser Stelle sei darauf hingewiesen, daß angenommen wird, daß nicht nur Zeiger auf Instanzen von *DRAAlgorithm* gespeichert werden, sondern auch die Instanzen selbst – also die ganzen in ihnen enthaltenen Daten. Auf welche Art das erfolgt, wird hier nicht betrachtet, da es ein Implementierungsdetail ist und vom verwendeten Backend-Datenhaltungssystem abhängt.

Nachfolgend ist eine Methode der Datenhaltung des Servers in Pseudocode wiedergegeben. Sie ist dafür da, um eine Instanz des Dead-Reckoning-Algorithmus zurückzuliefern, der die Informationen zu einem bestimmten Mobilien Objekt enthält.

```
DRAAlgorithm findEntry( objectID ) {
    return SELECT * FROM data WHERE OID=objectID;
```

---

```

}
```

Anmerkung: „findEntry“ soll den Zeiger auf das Objekt zurückgeben, daß die Daten enthält – nicht etwa eine Kopie vom Objekt erstellen! Änderungen werden also stets global sichtbar. Um gleichzeitiges Ändern zu verhindern ist ein Sperrmechanismus nötig!

Für eine konkrete Realisierung, ist eine Adaption der hier geforderten Basisfunktionalität auf die gewählte Technologie nötig. So ist es beispielsweise möglich, den Datenbestand in einem CORBA-Orb zu speichern oder eine relationale Datenbank als Backendsystem mit einer Zugriffskomponente zu verwenden. Die hier vorgestellten Lösungen sollen lediglich ein generisches Design aufzeichnen und so viele Freiheiten für eine tatsächliche Implementierung wie möglich übrig lassen. Unabhängig davon wird im folgenden der Begriff „Datenbank“ verwendet, um das Datenhaltungssystem zu bezeichnen.

## 4.5.2 Beantwortung von Lokationsanfragen

Wie bereits erwähnt, ist eine der beiden Grundaufgaben des Lokationsdienstes, die Beantwortung von Anfragen nach dem Aufenthaltsort eines bestimmten Objektes. Das jeweilige Objekt wird dabei über einen Bezeichner identifiziert. Diese werden folgendermaßen bearbeitet:

```

Position getCurrentPosition( objectID ) {
    myAlgorithm = findEntry( objectID );
    Pos = myAlgorithm.computeCurrPos();
    return Pos;
}
```

Zuerst wird der Eintrag zum jeweiligen Objekt aus der Datenbank herausgesucht. Anschließend dient die Methode „computeCurrPos“ dazu, anhand der übergebenen Update-Nachricht die aktuelle Soll-Position zu bestimmen. Da die Soll-Position vom verwendeten Algorithmus abhängt, muß die Methode zum jeweiligen Algorithmus gehörende Methode aufgerufen werden. (Das geschieht über den Mechanismus des Dynamischen Bindens!)

## 4.5.3 Verarbeitung von Updatenachrichten

Neben der Beantwortung von Positions-Anfragen, müssen auch eingehende Update-Nachrichten seitens der Mobilien Objekte verarbeitet werden. Die ankommende Update-Nachricht muß dazu in der Datenbank abgelegt werden. Da immer nur das zuletzt empfangene Update benötigt wird, wird das bereits in der Datenbank enthaltene Update einfach überschrieben. Das geschieht wie folgt:

```

computeUpdate( UpdateMessage ) {
    myAlgorithm = findEntry( UpdateMessage.objectID );
    UpdateMessage.timestamp = System.getCurrentTime(); (*)
    myAlgorithm.setNewUpdate( UpdateMessage );
}
```

In der Zeile (\*) wird die absolute Zeit, die das Update enthält, überschrieben und durch die momentane Systemzeit des Servers ersetzt. Das liegt daran, daß die Uhren vom Client und Server nicht synchron laufen. Eine Erläuterung dieses Sachverhalts ist im Kapitel 3.2.5 nachzulesen.

Der hier wiedergegebene Code setzt dabei die vereinfachte Formel ein, um die Zeit  $\Delta t$  zu berechnen. Die Kommunikationszeit  $T_{Komm}$  zwischen Client und Server findet also keine Beachtung. Möchte man diese auch berücksichtigen, müßte man `timestamp` wie folgt setzen:

```
UpdateMessage.timestamp = System.currentTimeMillis() - TKomm;
```

#### 4.5.4 Weitere Aufgaben des Servers

Neben den hier näher vorgestellten Aufgaben, Beantwortung von Lokationsanfragen und Verarbeitung von Updatenachrichten, ergeben sich darüber hinaus selbstverständlich noch weitere Aufgaben für den Lokationsserver. Dazu zählen Mechanismen zur Anmeldung bzw. Abmeldung eines neuen Mobilen Objektes, Sicherheitsvorkehrungen zur Zugriffskontrolle, aber auch zur Absicherung einer Nachricht (Verschlüsselung gegen Abhören; Zusicherung, daß Daten nicht verändert wurden und „Digitale Unterschrift“). Die Sicherheitsvorkehrungen müssen auch dazu dienen, Sabotage dritter auszuschließen (z.B. Replay von abgefangenen Nachrichten muß erkannt werden). Außerdem muß es in einem realen System auch Möglichkeiten geben, einen Dead-Reckoning-Algorithmus anfangs auszuhandeln und später dynamisch zu einem neuen überzugehen.

Diese weitergehenden Aufgaben sind sicherlich ein wichtiger Teil eines Lokationsdienstes, die bei einer konkreten Realisierung unbedingt berücksichtigt werden müssen. Sie sind aber nicht Gegenstand dieser Arbeit. Daher werden diese Aspekte nicht weiter vertieft, sondern wurden hier lediglich der Vollständigkeit halber erwähnt.

### 4.6 Vorschlag für ein Protokoll

In diesem Kapitel geht es darum, ein Protokoll vorzustellen, welches zum Ablauf eines Verfahrens benötigt wird. Dabei ist es nicht das Ziel, ein bis ins Detail ausgearbeitetes Protokoll wiederzugeben, bei dem sämtliche Nachrichten bis zum kleinsten Bit spezifiziert sind. Vielmehr sollen die benötigten Bestandteile des Protokolls aufgezählt und deren Funktion aufgezeigt werden. Das hier dargestellte Protokoll müßte dann detaillierter spezifiziert werden, um es in einem konkreten Fall zu implementieren. Insbesondere die Behandlung von Fehlerfällen wie Verbindungsabbrüche werden nur am Rande behandelt.

Eine Implementierung des Protokolls erfolgte aber nicht im Rahmen dieser Arbeit, sondern ist späteren Projekten vorbehalten!

#### 4.6.1 Bestandteile

Im folgenden werden vier Phasen des Kommunikationsprotokolls unterschieden: Verbindungsaufbau, stehende Verbindung, Neuaushandlung der Parameter und Verbindungsabbau. Wie der Name schon ausdrückt geht es in der ersten Phase, dem Verbindungsaufbau, darum, die Verbindung zwischen einem Mobilen Objekt und dem Server herzustellen. Hierbei müssen die Parameter der Algorithmen ausgehandelt werden. Dazu gehören beispielsweise die maximal tolerierte Abweichung zwischen Soll- und Ist-Position oder das zugrundeliegende Dead-Reckoning-Protokoll. Ist der Verbindungsaufbau abgeschlossen, steht die Verbindung. Ab hier sendet das Mobile Objekt Updatenachrichten an

den Server. Sind neue Rahmenbedingungen eingetreten, welche die bestehende Verbindung nicht mehr erfüllen kann, so müssen die Parameter der Verbindung neu ausgehandelt werden.

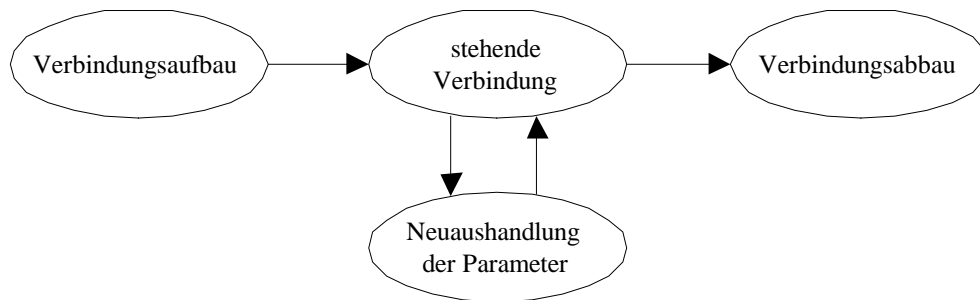


Abbildung 12 - Zustandsdiagramm der Phasen des Kommunikationsprotokolls

Im Prinzip geschieht hier das gleiche wie beim Verbindungsaufbau. Der Unterschied zum initialen Verbindungsaufbau ist aber, daß das Mobile Objekt dem Server bereits bekannt, da es sich schließlich angemeldet hatte. Bei der Neuaushandlung können zum einen die Parameter des gegenwärtigen Protokolls verändert werden (beispielsweise kann das Mobile Objekt einen größeren Toleranzabstand wählen) oder zu einem gänzlich anderen Algorithmus übergegangen werden (beispielsweise kann vom kartenbasierten Verfahren zum linearen Fortsetzen gewechselt werden, wenn der Bereich der Kartendaten verlassen wurde). Ist schließlich die Verbindung zum Server nicht mehr gewünscht, muß die Verbindung in der letzten Phase abgebaut werden.

Im Anschluß werden die einzelnen Phasen genauer betrachtet.

#### 4.6.2 Verbindungsaufbau

Zum Aufbau einer Verbindung werden mindestens die Nachrichten `ConnectRequest`, `ProtocolSelect` und `ProtocolParameter` benötigt. Im Idealfall läuft dann der Verbindungsaufbau ab wie in Abbildung 13.

Der Verbindungsaufbau startet durch die Initiative des Clients, indem dieser eine `ConnectRequest`-Nachricht an den Server schickt. In dieser Nachricht teilt er dem Server seine ID mit. Die ID muß auf irgendeine Weise eindeutig einem mobilen Gerät zugeordnet sein. Das kann beispielsweise dadurch erfolgen, daß die MAC-Adresse der Netzwerkkarte benutzt wird oder ein Client eine vorher festgelegte feste Nummer besitzt, die ihn identifiziert (vergleichbar mit einem Nummernschild bei einem Auto). Darüber hinaus gibt es noch eine Reihe von Algorithmen zur Erzeugung einer global eindeutigen ID. Die Eindeutigkeit der ID ist nötig, damit der Server die Clients unterscheiden kann.

Gegebenenfalls enthält die `ConnectRequest`-Nachricht auch Daten zur Authentifikation des Clients gegenüber dem Server, damit sich der Client seine Berechtigung nachweisen kann, sich mit dem gewählten Server zu verbinden. Dadurch ist es möglich, eine Zugriffskontrolle zu implementieren. Ist der Server bereit, eine Verbindung zu diesem Client aufzubauen, schickt er eine `Accept`-Nachricht zurück. Wird die Verbindung abgelehnt, erhält der Client eine `Disconnect`-Nachricht als Antwort.

Anschließend wählt der Client das Verfahren, nach dem die Vorhersagen der Soll-Positionen erfolgen soll. Da Client und Server das gleiche Verfahren anwenden müssen, ist es unerlässlich sich auf ein gemeinsames Verfahren zu einigen. Ist der Server mit der Wahl des Clients einverstanden, dann schickt er wieder eine `Accept`-Nachricht als Antwort zurück. Es kann aber auch vorkommen, daß der Server das vom Client vorgeschlagene Vorhersageverfahren nicht anbietet. Dann antwortet der Server mit einer `AvailablePortocols`-Nachricht. Darin listet er alle Verfahren auf, die er bereit ist, zu akzeptieren.

Der Client muß sich dann ein Verfahren daraus aussuchen und erneut eine `ProtocolSelect`-Nachricht zurückschicken, um den Server von seiner Wahl zu informieren. Der Server bestätigt die Wahl dann mit einer `Accept`-Nachricht. Kann oder

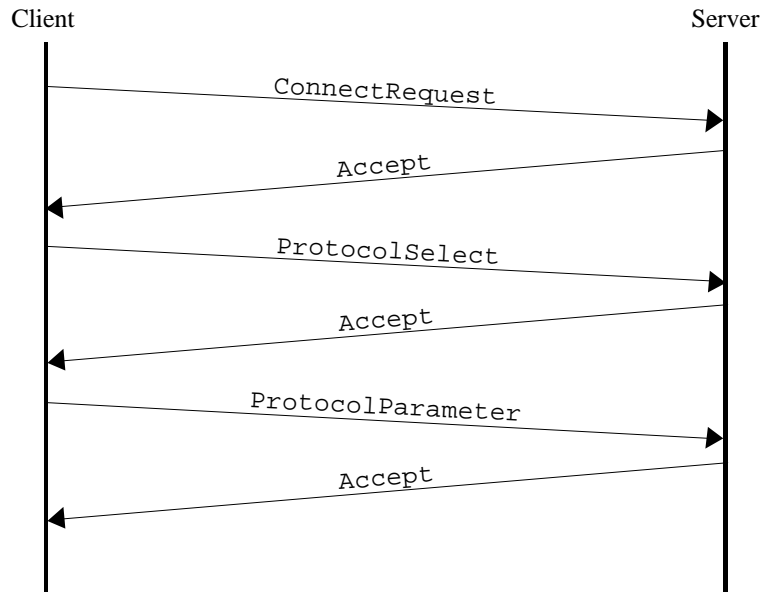


Abbildung 13 - Nachrichtenaustausch zum Aufbau einer Verbindung im Idealfall

möchte der Client dagegen keines der vom Server angebotenen Verfahren benutzen, muß er die Verbindung trennen. Dazu schickt er eine `Disconnect`-Nachricht an den Server, um diesen zu signalisieren, daß er den Verbindungsaufbau abgebrochen hat.

Nach der Festlegung des Verfahrens müssen noch die Parameter, also die Einstellungen des Verfahrens ausgehandelt werden. Hierzu dient die `ProtocolParameter`-Nachricht. Sie ist für jedes Verfahren spezifisch, da sich die Parameter von Verfahren zu Verfahren unterscheiden können. Beispielsweise benötigt das Verfahren Lineares Fortsetzen nur den Parameter maximaler Ungenauigkeitsradius. Bei einem Kartenbasierten Verfahren müßte darüber hinaus noch mindestens die Karte ausgehandelt werden, die eingesetzt wird. Wegen der unterschiedlichen Parameter je nach Verfahren, ist es einfacher, zuerst das Verfahren festzulegen (`ProtocolSelect`) und danach die jeweiligen Parameter vorzuschlagen. Durch diese Aufteilung ist es auf einfache Weise (mit wenig Kommunikationsvolumen) möglich, während eine Verbindung steht, die Parameter zu verändern.

Auf die `ProtocolParameter`-Nachricht reagiert der Server erneut mit einem `Accept`,

wenn er die Parameter akzeptiert. Ansonsten schickt er eine AvailableParameter-Nachricht zurück, in der er dem Client mitteilt, welche Parameter möglich sind. Im allgemeinen wird dies nur bei den kartenbasierten Verfahren erforderlich sein. Das liegt daran, daß es hier vorkommen kann, daß der Client eine Karte vorschlägt, die dem Server nicht bekannt ist.

Auf die AvailableParameter-Nachricht kann der Client analog zur AvailableProtocols-Nachricht entweder durch Sendung einer ProtocolParameter-Nachricht mit einem der Parameter (z.B. Karten) aus der Liste des Servers reagieren oder eine Disconnect-Nachricht verschicken, um die Verbindung abzulehnen. Da es gerade bei den kartenbasierten Verfahren vorkommen kann, daß der Verbindungsaufbau daran scheitert, daß sich Client und Server nicht auf eine gemeinsame Karte einigen können (weil beide verschiedene Karten besitzen), kann der Client den gesamten Verbindungsaufbau von vorn beginnen, indem er wieder eine ConnectRequest-

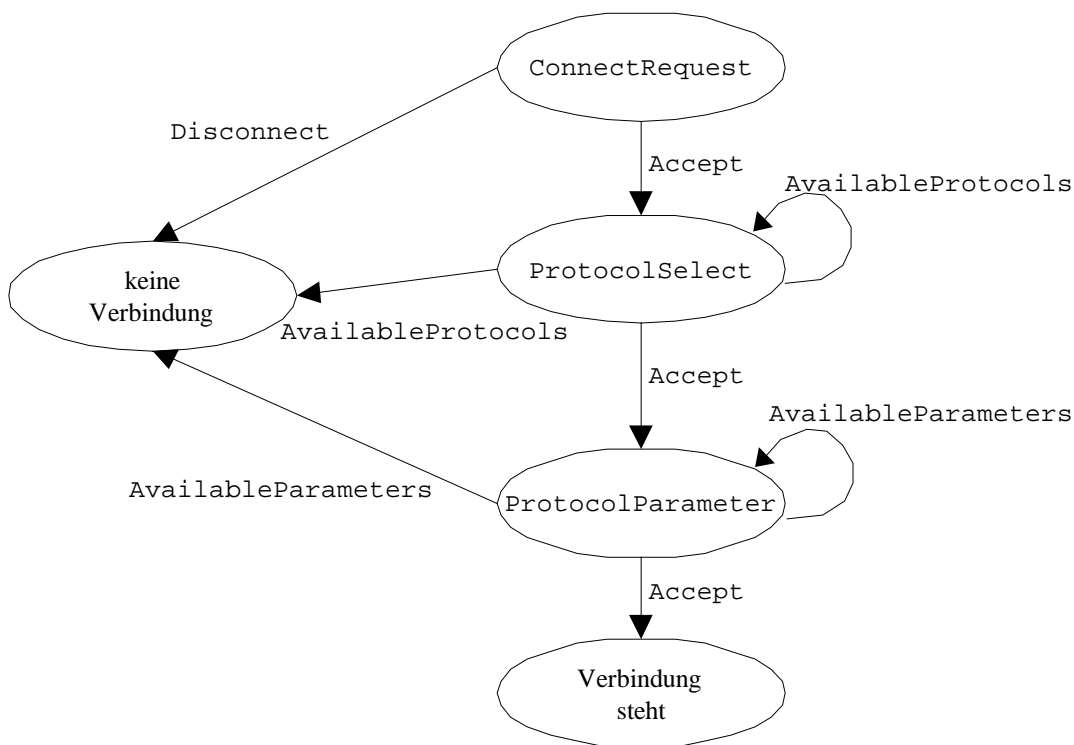


Abbildung 14 - Nachrichten beim Verbindungsaufbau

Nachricht schickt und dieses Mal ein nicht-kartenbasiertes Verfahren auswählt!

Hat der Server die ProtocolParameter-Nachricht mit einem Accept quittiert, steht die Verbindung. Der Client kann im folgenden Updatenachrichten verschicken.

Abbildung 14 zeigt den Verbindungsaufbau in einem Diagramm. Darin sind die Nachrichten des Clients innerhalb der Ellipsen dargestellt. Die Antworten des Servers sind als Kantenbeschriftungen aufgetragen und markieren so den Zustandsübergang als Reaktion auf die Nachrichten.

### 4.6.3 Während eine Verbindung besteht

Während die Verbindung steht, verschickt das Mobile Objekt bei Bedarf Updatenachrichten an den Server. Der Inhalt der Updates ist wie im Kapitel 4.1 (Seite 32) beschrieben. Den Empfang eines Updates bestätigt der Server mit einer `Acknowledge`-Nachricht.

Bleibt die `Acknowledge`-Nachricht aus (d.h. ein Timeout verstreicht), muß der Client davon ausgehen, daß die Verbindung zum Server unterbrochen ist. Darauf hin wird nicht die alte Nachricht einfach wiederholt, sondern eine neue (aktualisierte) Updatenachricht generiert, welche die neuesten Werte enthält.

Das ist ein wichtiger Unterschied zu den üblichen Protokollen zur Datenübertragung, bei denen es entweder wichtig ist, daß kein Datum verloren geht (ggf. erfolgt Retransmit) oder bei denen es unerheblich ist, wenn ein Datum verloren geht (Echtzeitdatenübertragung wie Videoübertragung), weil es zum Zeitpunkt des Retransmits veraltet und unbrauchbar wäre. Hier ist es zum einen wichtig, daß eine Updatenachricht beim Server ankommt (der Client muß sich darauf verlassen können), andererseits macht es keinen Sinn, ein veraltetes Update per Retransmit erneut zu schicken, weil die Daten zum Retransmit nicht mehr aktuell sind. Statt dessen muß die Anwendung – also der Dead-Reckoning-Algorithmus – ein neues (aktuelles) Update generieren und verschicken. Dazu muß es Feedback erhalten, ob eine gesendete Nachricht angekommen (`Acknowledge`) ist oder eine neue Nachricht verschickt werden muß (Ausbleiben von `Acknowledge` über einen bestimmten Zeitraum).

### 4.6.4 Neuaushandlung der Parameter

Wenn eine Verbindung zum Server steht, ist es möglich, sowohl das verwendete Verfahren zu wechseln als auch lediglich die Parameter neu festzulegen. Analog zum Verbindungsaufbau werden dazu die beiden Nachrichten `ProtocolSelect` und `ParameterChoice` verwendet. Der Server antwortet auch entsprechend mit `Accept` oder `AvailableProtocols` bzw. `AvailableParameter`. Näheres dazu ist im Kapitel 4.6.2 (siehe oben) nachzulesen.

Um zu einem gänzlich neuen Verfahren zu wechseln, muß der Client zunächst `ProtocolSelect` und anschließend `ParameterChoice` senden. Um lediglich die Parameter für das gegenwärtig benutzte Verfahren zu verändern, ist es auch möglich nur die `ParameterChoice`-Nachricht allein zu verschicken und so den Kommunikationsaufwand möglichst gering zu halten. Das neue Verfahren bzw. die neue Einstellung wird erst dann übernommen, wenn der Server die Nachricht `ProtocolParameter` mit `Accept` quittiert hat. Solange das nicht der Fall ist, gilt weiterhin das alte Verfahren! Insbesondere folgt hieraus, daß das `Accept` nach der `ProtocolSelect`-Nachricht (noch) nicht zum Wechsel des Verfahrens führt!

Benötigt wird die Möglichkeit, das Verfahren oder die Parameter zu verändern, beispielsweise beim kartenbasierten Verfahren, wenn das Mobile Objekt den Kartenbereich verläßt und eine neue Karte ausgewählt werden muß oder zu einem anderen Verfahren übergegangen werden muß. Ein anderes Szenario, wäre beispielsweise, wenn das Mobile Objekt einen bestimmten Bereich verläßt und es nicht mehr so genau lokalisiert werden muß. Dann kann man den Toleranzradius erhöhen und spart sich Kommunikation, weil weniger

Updates verschickt werden müssen. Wird später wieder eine feinere Auflösung des Aufenthaltsortes benötigt, kann dann der Toleranzradius wieder verkleinert werden.

### **4.6.5 Ende der Verbindung**

Die wohl einfachste Nachricht ist die Nachricht zum Beenden der Verbindung: `Disconnect`. Diese Nachricht wird immer vom Client an den Server gesendet. Lediglich beim Auftreten von Fehlern, könnte es Erforderlich sein, daß der Server die Verbindung seinerseits aktiv abbricht.

Empfängt der Server eine Ende-Nachricht, kann er alle Daten „vergessen“, die er zum entsprechenden Mobilen Objekt gespeichert hatte. Das bedeutet, daß er die ID des Mobilen Objektes nicht mehr zu speichern braucht und auch sämtliche Zustandsinformationen wie die letzte empfangene Position des Clients und die Daten des Dead-Reckoning-Protokolls verworfen werden können. Sollten nach Abbau der Verbindung noch Nachrichten vom Client empfangen werden (aus welchen Gründen auch immer), braucht der Server nicht mehr auf diese zu reagieren, sondern kann sie ignorieren. Die einzige Ausnahme stellt selbstverständlich die Nachricht zum Aufbau einer Verbindung dar, sonst wäre es einem Client nie wieder möglich, sich mit einem Server zu verbinden, von dem es sich irgendwann einmal getrennt hatte!

## 5 Lineares Fortsetzen

---

Beim Linearen Fortsetzen handelt es sich um das erste Verfahren, das näher beschrieben werden soll. Dabei werden ausgehend von einer vom Lagemeßsystem bestimmten aktuellen Ist-Position und der Geschwindigkeit die Soll-Positionen (d.h. die zukünftigen Aufenthaltsorte) berechnet. Zur Funktion des Verfahrens sind darüber hinaus keine Zusatzdaten nötig. In der hier präsentierten Lösung wird stets davon ausgegangen, daß sich das Mobile Objekt von der gegenwärtigen Position gerade (also linear) in die gleiche Richtung und mit gleicher Geschwindigkeit weiterbewegt wie bisher. Es wäre aber auch ohne Auswirkung auf das Arbeitsprinzip des Verfahrens leicht möglich, beispielsweise eine Maximal- oder Minimalgeschwindigkeit des Objektes zu benutzen.

Ist das Lagemeßsystem in der Lage, eine Orientierung zu bestimmen, kann diese Orientierung als Richtungsvektor für die weitere Bewegung benutzt werden. Ansonsten muß die Richtung aus den letzten gemessenen Ist-Positionen errechnet werden. Im einfachsten Fall wird der Richtungsvektor aus der momentanen Position und der zuletzt gemessenen Position errechnet. Auf diese Weise kann man auch die Geschwindigkeit berechnen, ohne ein speziellen Geschwindigkeitsmesser einzusetzen.

Um die Ungenauigkeiten des Lagemeßsystems möglichst zu minimieren, kann man die Bewegungsrichtung nicht nur aus einem Meßpunkt bestimmen, sondern aus mehreren. Aus den Ist-Positionen bestimmt man dazu eine Regressionsgerade, die als Bewegungsrichtung dient.

Die Bestimmung einer Regressionsgeraden wird im sich gleich anschließenden Kapitel 5.1 beschrieben. Das Kapitel 5.2 beschäftigt sich dann mit der Implementierung des Algorithmus für das Lineare Fortsetzen.

### 5.1 Linienglättung

Die Linienglättung ist ein Ansatz, der durch Einbeziehung mehrerer Lagemeßpunkte versucht, eine möglichst gute Regressionsgerade zu errechnen. Der Grund für den Aufwand ist der, daß die Lagemessung mit Meßfehlern behaftet ist. Je nach zur Verfügung stehenden Meßsystem ist die mögliche Abweichung des bestimmten Lagepunktes von der tatsächlichen Lage unterschiedlich. Beispielsweise hat DGPS eine Abweichung von 2-5m.

Bei der Bestimmung der Position mit DGPS für die Simulation (siehe Kapitel 7), haben wir gelegentlich Aussetzer (d.h. kurzzeitig keine Positionsmeldung) und Wiederholungen von Positionen durch das GPS-Gerät beobachtet. Diese Unstetigkeiten können durch mit Hilfe von Linienglättung im gewissen Umfang geglättet werden.

## 5.1.1 Mathematische Grundlagen

Bevor es an die Entwicklung eines Glättungsalgorithmus gehen kann, müssen zunächst die mathematischen Grundlagen bereitgestellt werden. Das später in diesem Kapitel vorgestellte Glättungsverfahren basiert auf dem Verfahren zur Berechnung einer Regressionsgeraden nach Gauß. Ausgehend vom Gauß-Verfahren, das aus einer Anzahl von gegebenen Punkten eine Geradengleichung der Form  $y=mx+c$  bestimmt, wird ein Verfahren entwickelt, das mit Vektoren arbeitet, um beliebig orientierte Regressionsgeraden bestimmen zu können.

Grundvoraussetzung für die Funktion des hier vorgestellten Regressionsalgorithmus ist die Verwendung eines karthesischen Koordinatensystems. Folglich ist die Anwendung des Geodetischen Koordinatensystems wie WGS84 von GPS nicht geeignet, da es keine orthogonalen Koordinatenachsen besitzt. Daher müssen die Koordinateninformationen des GPS-Gerätes in ein karthesisches Koordinatensystem wie UTM oder Gauß-Krüger überführt werden, um korrekte Resultate zu bekommen.

Die Transformation von WGS84 nach Gauß-Krüger besteht aus einer Reihe komplexer Berechnungsschritte, welche in NEXUS bereits implementiert sind.

### 5.1.1.1 Gaußverfahren

Zunächst wird das original Gaußverfahren erläutert, welches wie bereits erwähnt dann erweitert wird, um unseren Erfordernissen zu genügen. Das Verfahren dient zur Bestimmung einer Regressionsgeraden anhand von vorgegebenen Punkten. Die Parameter der Geraden sind so zu bestimmen, so daß gilt, daß die Summe aller quadrierten Abstände der Punkte zur Geraden minimal wird.

Sei  $f(x)=mx+c$  die gesuchte Geradengleichung. Dabei ist  $m$  die Steigung der Geraden und  $c$ , der sogenannte  $y$ -Achsenabschnitt. Sei  $d_i$  der Abstand des Punktes  $P_i(x_i/y_i)$  von der Geraden wie folgt definiert:

$$d_i = m \cdot x_i + c - y_i \quad (1)$$

Das entspricht dem parallel zur  $y$ -Achse gemessenen Abstand zwischen dem Punkt und der Geraden. (In Abbildung 15 ist das der Abstand  $d_i$ .)

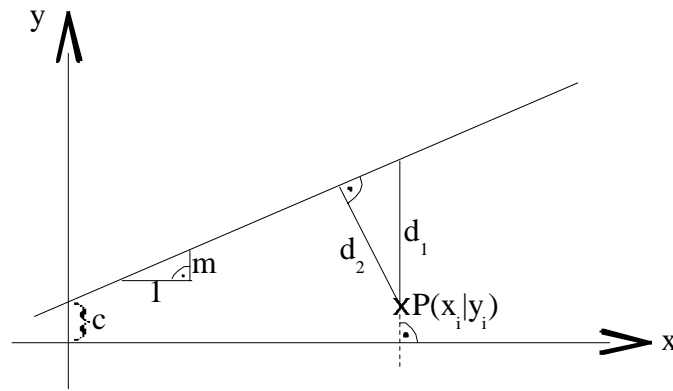


Abbildung 15 – Verdeutlichung der Unterschiede bei der Abstandsmessung. Abstand  $d_1$  entspricht dem Abstand wie er beim Gaußverfahren gemessen wird. Abstand  $d_2$  zeigt dagegen den lotrecht gemessenen Abstand.

Um die Gerade zu finden, sind die Werte  $m$  und  $c$  gesucht, für die die Funktion  $g(m,c)$  ihr Minimum erreicht. Dabei seien die Punkte  $P_i$  von 0 bis  $n$  nummeriert.

$$g(m,c) = \sum_{i=0}^n d_i \quad (2)$$

Durch Einsetzen von (1) in (2), anschließend Ableiten nach  $m$  und  $c$  zur Bestimmung des Minimums ergibt sich folgende Lösung:

$$m = \frac{Z}{N} = \frac{\sum_{i=0}^n (x_i \cdot y_i) - \frac{1}{n} \cdot \sum_{i=0}^n x_i \cdot \sum_{i=0}^n y_i}{\sum_{i=0}^n x_i^2 - \frac{1}{n} \cdot \left( \sum_{i=0}^n x_i \right)^2} \quad (3)$$

$$c = \frac{1}{n} \cdot \left( \sum_{i=0}^n y_i - m \cdot \sum_{i=0}^n x_i \right) \quad (4)$$

Anhand der obigen beiden Gleichungen sind dann die beiden Parameter  $m$  und  $c$  der Gerade bekannt. Die gesuchte Gerade ergibt sich dann zu  $y = mx + c$ .

### 5.1.1.2 Geraden in Vektordarstellung

Um von der algebraischen Darstellung des Gaußverfahren zur Vektordarstellung überzugehen, müssen zunächst ein paar Grundlagen der Vektordarstellung eingeführt werden. Die hier eingeführten Begriffe dienen dann nachfolgend als Bezugsgrundlage für die Beschreibung des Vektoralgorithmus.

Allgemeine Vektorgleichung der Geraden:

$$\vec{y} = \vec{m} \cdot t + \vec{c}$$

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} \cdot t + \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

Dabei ist  $\vec{m}$  der Richtungsvektor und  $\vec{c}$  der Stützpunkt der Geraden.

Bei bekanntem Richtungswinkel  $\alpha$  ergibt sich die Geradengleichung zu:

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot t + \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \text{ bei } \alpha = 90^\circ \text{ oder } \alpha = 270^\circ$$

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \tan \alpha \end{pmatrix} \cdot t + \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \text{ sonst}$$

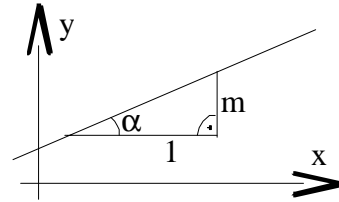


Abbildung 16 - Veranschaulichung des Zusammenhangs zwischen Winkel und Steigung

Dabei entspricht ein Richtungswinkel von  $\alpha=90^\circ$  bzw.  $\alpha=270^\circ$  einer Geraden, die parallel zur y-Achse verläuft.

### 5.1.1.3 Verallgemeinerung des Verfahrens

Das vorgestellte Gaußverfahren, das die Summe der quadratischen Abstände minimiert, hat zwei Nachteile. Zum einen mißt es den Abstand eines Punktes zur Regressionsgeraden nicht senkrecht, sondern nur in y-Richtung (siehe Abbildung 15, Seite 47), und zum anderen scheidet es bei Geraden, die parallel zur y-Achse verlaufen.

Der zweite Nachteil liegt nicht im Verfahren begründet, sondern in der zugrundeliegenden Mathematik, weil man mit der allgemeinen Geradengleichung  $y=mx+c$  keine Parallelen zur y-Achse darstellen kann. (Je steiler die Gerade wird, desto mehr strebt  $m$  gegen  $+\infty$  oder  $-\infty$ .) Daher ist die naheliegende Idee, das Verfahren mittels Vektoren durchzuführen. Dadurch wäre es möglich beliebig orientierte Geraden zu berechnen.

Auch der erste Nachteil, daß Entfernungen nicht senkrecht gemessen werden, wäre behebbbar, indem die Abstände  $d_i$  stets lotrecht gemessen werden<sup>7</sup>. Im folgenden soll das Prinzip mit Vektoren näher beschrieben werden.

Ein zu betrachtender Meßpunkt in Vektordarstellung sei gegeben durch:

$$\vec{p}_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

Der senkrecht gemessene Abstand zwischen dem Meßpunkt und der zu bestimmenden Geraden ergibt sich wie folgt (nach [MERZ94]):

$$D^2 = \frac{[m_1(y_i - c_2) - m_2(x_i - c_1)]^2}{m_1^2 + m_2^2}$$

<sup>7</sup> Es besteht kein linearer Zusammenhang zwischen dem lotrechten Abstand und dem Abstand  $\Delta y$  wie er im Gauß-Verfahren benutzt wird. Daher sind verschiedene Lösungen für die Gerade bei unterschiedlicher Abstandsmessung zu erwarten!

Analog zum vorgestellten Gaußverfahren ist die Gerade gesucht, für die gilt:

$$f(m_1, m_2, c_1, c_2) = \sum_{i=0}^n D^2 = \text{minimal} \quad (3)$$

Um das Minimum zu finden, müssen die partiellen Ableitungen der Gleichung (3) gebildet werden, auf Null gesetzt werden und anschließend die Lösung dieses Gleichungssystems bestimmt werden. Die Bestimmung der vier Ableitungen ist dabei unproblematisch. Man bekommt allerdings recht komplexe Gleichungen als Lösung.

Analog zum vorgestellten Gauß-Verfahren wäre der nächste Schritt, diese vier Ableitungen auf Null zu setzen und das so erhaltene Gleichungssystem zu lösen. Die daraus resultierende Lösung für das Gleichungssystem ist allerdings sehr lang<sup>8</sup>. Die direkte Berechnung einer Geraden bei vorgegebenen Punkten durch einen Algorithmus ist zu komplex und die Übertragung der vielen Lösungsterme in Programmcode wäre sehr fehleranfällig und nicht wartbar. (Fehler wären nur sehr schwer zu finden)

Daher wurde in der Implementierung darauf verzichtet, eine direkte Lösung zu verwirklichen. Statt dessen ist die nachfolgend vorgestellte Iterationslösung umgesetzt worden.

### 5.1.2 Iterationslösung

Die nachfolgend vorgestellte Iterationslösung bedient sich einerseits des Gauß-Verfahrens zur Bestimmung der Regressionsgeraden, eliminiert aber andererseits dessen bereits erwähnte Nachteile (siehe dazu Kapitel 5.1.1.3).

Der Kerngedanke des Algorithmus ist, daß das Gauß-Verfahren nur dann die Abstände lotrecht zur Regressionsgeraden mißt, wenn die gefundene Gerade waagrecht ist. (d.h. Steigung  $m=0$ ) Um das zu erreichen, müssen die Meßpunkte, die zur Berechnung herangezogen werden, durch eine affine Drehung um einen Winkel  $\alpha$  soweit rotiert werden bis die berechnete Gerade waagrecht liegt. Anhand des Drehwinkels kann man dann die Steigung der Geraden vor der affinen Abbildung berechnen.

Da der Winkel  $\alpha$  a priori nicht bekannt ist, muß man ihn iterativ bestimmen. Dazu wendet man das Gauß-Verfahren zunächst im ungedrehten Fall an ( $\alpha=0^\circ$ ). Man bestimmt den Winkel  $\beta$ , den die gefundene Gerade mit der x-Achse bildet. Der neue Drehwinkel ergibt sich zu  $\alpha' = \alpha - \beta$ . Man dreht die Meßpunkte um den neuen Winkel  $\alpha$  und bestimmt  $\beta$  erneut.

Die Iteration kann abgebrochen werden, wenn gilt  $\beta \leq \epsilon$ , wobei  $\epsilon$  die vorgegebene Genauigkeit ist.

**Algorithmus<sup>9</sup>:**

1.  $\alpha = 0^\circ$
2. Berechne Nenner  $N$  gemäß Formel (3) aus Kapitel 5.1.1.1  
*Achtung:* Sollte die Gerade senkrecht sein, würde das normale Gaußverfahren scheitern. Bei einer senkrechten Geraden, nimmt der Nenner  $N$  der Formel zur Berechnung der Geradensteigung  $m$  den Wert 0 an.

<sup>8</sup> Die Lösung umfaßte mehrere Bildschirmseiten bei der Ausgabe durch ein Algebraprogramm.

<sup>9</sup> Anmerkung: Winkel werden im Gegenuhrzeigersinn gemessen – wie in der Mathematik üblich

3. Ist ( $N=0$ ) dann
  - Drehe Punktmenge um Winkel  $270^\circ$  um den Ursprung
4. Bestimme Gerade nach (normalen) Gauß'schen Regressionsverfahren (wie in Kapitel 5.1.1.1)
5. Bestimme deren Winkel  $\beta$  (Winkel zwischen Regressionsgeraden und x-Achse)
6. Ist ( $|\beta| < \epsilon$ ) (d.h. ist die Gerade waagrecht im Rahmen der gewünschten Genauigkeit?)
 

Nein:

  - $\alpha = \alpha - \beta$
  - Drehe Punktmenge um Winkel  $\alpha$  um den Ursprung
  - Wiederhole ab Punkt 2.

Ja: fertig!

  - Rückgängigmachung Abbildung der zuletzt gefundenen Geraden durch affine Drehung um  $-\alpha$  um den Ursprung des Koordinatensystems.

### 5.1.3 Optimierung des Änderungsverhaltes des Richtungsvektors

Für die Dead-Reckoning-Verfahren ist es von größter Bedeutung neben dem momentanen Aufenthaltsort auch eine Bewegungsrichtung zu kennen, um Vorhersagen über das zukünftige Verhalten zu treffen. Ein Richtungsvektor kann bereits ab 2 Positions-Meßpunkten ermittelt werden. Mit steigender Anzahl der herangezogenen Punkte, steigt auch die Genauigkeit, mit der die Richtung errechnet werden kann, weil durch die steigende Anzahl der Punkte einzelne Meßfehler das Ergebnis weniger stark beeinflussen.

Dabei gilt es aber abzuwägen, welche Punktzahl die besten Ergebnisse liefert. Der Grund ist, daß der durch Mittelung bestimmte Richtungsvektor immer träger auf Richtungsänderungen (Kurvenfahren oder Richtungsumkehr) reagiert, desto mehr zurückliegende Punkte berücksichtigt werden.

In Anlehnung an den Douglas-Peucker-Algorithmus zur Glättung von Polygonzügen, wird nachfolgend ein Algorithmus vorgestellt, der Richtungsänderungen erkennen soll.

#### Algorithmus:

1. Bestimme Regressionsgerade zu den bekannten Punkten
2. Lege Toleranzband um die Gerade
3. Sind alle Punkte innerhalb des Toleranzbandes?
  - JA:  
Verwende Richtungsvektor der Regressionsgeraden als Orientierung und beende Algorithmus
  - NEIN:  
Finde Punkt  $P^*$ , der am weitesten von der Geraden entfernt ist.

Verwerfe alle Punkte, die (zeitlich) vor  $P^*$  liegen  
wiederhole ab 1.

Die beiden Abbildungen 17 und 18 verdeutlichen die Auswirkung des Einsatzes des obigen Algorithmus.

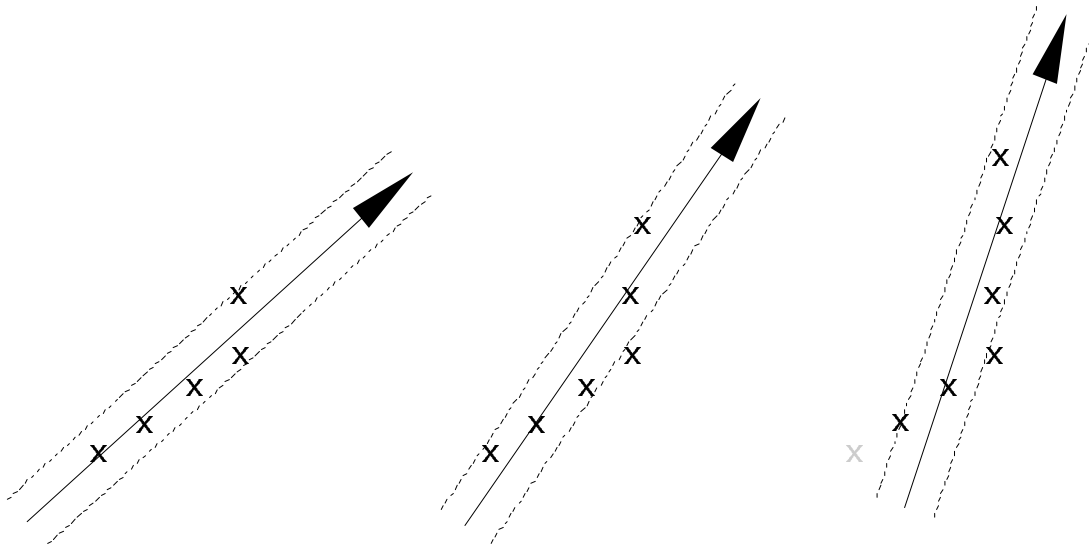


Abbildung 17 – Veränderung des Richtungsvektors, unter Verwendung von 6 Punkten zur Bestimmung der Regressionsgerade. Man kann erkennen, daß der Richtungsvektor nur sehr träge auf die Richtungsänderung reagiert!

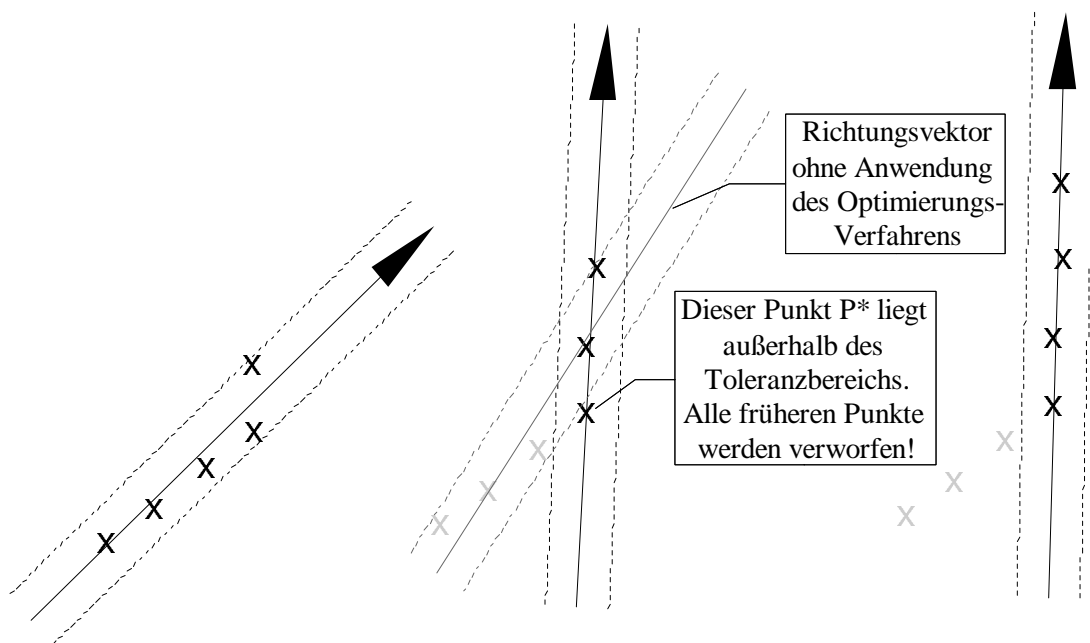


Abbildung 18 – Veränderung des Richtungsvektors (wieder maximal 6 Punkte zur Bestimmung der Regressionsgeraden). Dieses Mal kommt der oben vorgestellte Algorithmus zum Zuge. Der Richtungsvektor folgt schnell der Richtungsänderung.

## 5.2 Ablauf des Verfahrens

Bisher wurden nur die mathematischen Grundlagen zur Linienglättung dargestellt. Nun soll der gesamte Ablauf des Algorithmus zur linearen Fortsetzung zusammengefaßt werden. Dazu werden die Methoden der Klasse *DRLinear* im Pseudocode wiedergegeben und beschrieben. Die grundlegenden Erklärungen, wann welche Methode aufgerufen wird, sind im Kapitel 4 (Seite 32ff) nachzulesen. Besonders wird auf das Kapitel 4.2 (Seite 33) verwiesen, in dem die abstrakte Grundklasse *DRAlgorithm* beschrieben wird, da *DRLinear* eine Subklasse von *DRAlgorithm* ist.

Bevor es an die Erläuterung der einzelnen Methoden gehen kann, wird der Kopf der Klasse abgebildet, um das Verständnis zu ermöglichen, wenn Bezüge auf die Attribute der Klasse gemacht werden.

```
class DRLinear extends DRAlgorithm {
    OID          myOID;
    double       maxDev;
    int          usedPoints;

    Position[]  lastPos;
    long[]      lastTimes;
    PosUpdate   currUpdate;
    [...]
}
```

Ab hier werden zunächst die Methoden als Code wiedergegeben und anschließend erklärt.

```
init( theOID, double maximumDev, int usedPts ) {
    myOID      = theOID;
    maxDev     = maximumDev;
    usedPoints = usedPts;
    max = 0;
    min = 0;
}
```

*init*: Diese Methode dient lediglich zur Initialisierung des Zustandes des Objektes. Dabei werden auch die Parameter des Verfahrens festgelegt. Der Parameter „maximumDev“ legt den Toleranzradius fest, der beim Vergleich der Abweichung zwischen Soll- und Ist-Position verwendet wird. Der Parameter „usedPts“ bestimmt, wie viele Lagepunkte für die Berechnung der Regressionsgeraden herangezogen werden sollen.

```
Position correctCurrPos( Position P ) {
    // no correction needed!
    // but: store P in Position set!
    max=max+1;
    lastPos[max] = P;
    lastTimes[max] = current_Time();
    if ( max-min>usedPoints ) {
        min=min+1;
    }
    return P;
}
```

*correctCurrPos*: Bei der linearen Fortsetzung wird keine Korrektur der vom Meßsystem

bestimmten Ist-Lage durchgeführt. Daher ist der Rückgabewert der Methode gleich dem übergebenen Parameter. Statt dessen wird die im Parameter übergebene Position und der Zeitstempel intern gespeichert. Diese Daten werden für die Funktion des Algorithmus benötigt.

```

Position computeCurrPos() {
    if ( currUpdate!=null ) {
        // compute distance from position of last update
        deltaT = lastTimes[max] - currUpdate.timestamp;
        deltaS = currUpdate.v * deltaT;
        expected_P = "lastPos[max]+deltaS with right angle";
        return expected_P;
    } else {
        // here: no update to work with!
        return lastPos[max];
    }
}

```

*computeCurrPos*: Berechnet die Soll-Position anhand des letzten Updates. Dazu wird die verstrichene Zeit zwischen dem Zeitstempel des letzten Updates und dem Zeitpunkt berechnet, an welchen in der Methode *correctCurrPos* eine neue Ist-Position gespeichert wurde. Aus dieser Zeitdifferenz wird die Strecke berechnet, die das Mobile Objekt zurückgelegt hat, wenn die Geschwindigkeit des letzten Updates exakt eingehalten wurde. Um schließlich die Soll-Position zu berechnen muß diese Strecke in Richtung des Winkels aus dem letzten Update zu der Position des Updates addiert werden. Ist noch kein Update verschickt worden, kann die ganze Berechnung nicht stattfinden. Daher wird der gerade erst gespeicherte Punkt (*lastPos[max]*) als Soll-Position zurückgegeben.

```

boolean deviationExceeded() {
    if (Distance(lastPos[max],computeCurrPos())>maxDev) OR
        (currUpdate==null) {
        return true;
    } else {
        return false;
    }
}

```

*deviationExceeded*: Überprüft, ob die Ist-Position (*lastPos[max]*) weiter als der Toleranzradius von der Soll-Position entfernt ist. Der Rückgabewert ist „true“, wenn der Maximalabstand überschritten ist. Da es unmittelbar nach der Initialisierung noch keine Update-Nachricht gibt, um die Soll-Position sinnvoll zu berechnen, wird dieser Fall abgefragt (*currUpdate==null*). In einem solchen Fall, muß die erste Update-Nachricht an den Server geschickt werden.

```

PosUpdate computeNewUpdate() {
    if ( currUpdate!=null ) {
        VectorLine resultLine;
        resultLine = Regression line computed with points
                        lastPoints[min] to lastPoints[max];
        v = Distance(lastPos[max],lastPos[min])/
                (lastTimes[max]-lastTimes[min]);
    }
}

```

```

        currUpdate = new PosUpdate( myOID,
                                    lastTimes[max],
                                    lastPos[max],
                                    resultLine.angle,
                                    v );
    } else {
        currUpdate = new PosUpdate( myOID,
                                    lastTimes[max],
                                    lastPos[max],
                                    0,
                                    0 );
    }
    return currUpdate;
}

```

*computeNewUpdate*: Zur Berechnung einer neuen Updatenachricht wird zunächst aus der bei der Initialisierung festgelegten Anzahl von Stützpunkten eine Regressionsgerade berechnet. Die Berechnung wurde bereits im vorherigen Kapitel 5.1 dargestellt. Als Geschwindigkeitsvorhersage erhält das Update die Durchschnittsgeschwindigkeit auf der Strecke der letzten Ist-Positionen. Für die Geschwindigkeitsberechnung wird die gleiche Anzahl an Punkten wie bei der Berechnung der Regressionsgeraden herangezogen. Handelt es sich um das erste Mal, daß *computeNewUpdate* aufgerufen wurde, ist *currUpdate* nicht initialisiert. Da zu diesem Zeitpunkt lediglich eine einzelne Ist-Position erfaßt wurde, kann keine vollständige Updatenachricht erstellt werden, weil weder Richtung noch Geschwindigkeit berechnet werden können. Statt dessen wird eine Updatenachricht zurückgeliefert, bei der Richtung und Geschwindigkeit den Betrag 0 zugewiesen bekommen. Beim nächsten Mal, wenn *computeNewUpdate* aufgerufen wird, kann dann eine aussagefähigere Nachricht zusammengestellt werden.

```

setNewUpdate( PosUpdate newUp ) {
    currUpdate = newUp;
}

```

*setNewUpdate*: Diese Methode benötigt der Server, um dem Objekt ein vom Client empfangene Updatenachricht mitzuteilen. Erst dadurch ist es der Instanz möglich, Soll-Positionen zu berechnen, da der Server keine Ist-Positionen kennt, was Voraussetzung wäre, um alle Daten selbst zu berechnen.

Die Performance des hier vorgestellten Algorithmus wurde durch Simulation gemessen und ist zusammen mit den Ergebnissen für den kartenbasierten Ansatz im Kapitel 7 ab Seite 80 zu finden.

## 6 Kartenbasierte Koppelnavigation

---

Nach der Darstellung des Verfahrens zum Linearen Fortsetzen im vorangegangenen Kapitel, geht es hier um das zweite vertiefte Verfahren, die Kartenbasierte Koppelnavigation. Elemente des Linearen Fortsetzens werden auch bei diesem Verfahren auftauchen und dem Leser daher bekannt vorkommen. Wenn dies der Fall ist, werden aber keine Details wiederholt, da sie im Kapitel 5 (ab Seite 45) nachgelesen werden können.

Kernbestandteil des Kartenbasierten Verfahrens sind die elektronischen Karten. Daher geht es zunächst um die Datenstruktur für solche Karten. Durch den Einsatz von Karten ist es möglich, Updatenachrichten einzusparen, weil die Streckenverläufe – insbesondere bei Kurven – vorher bekannt sind, was ein wesentlicher Vorteil gegenüber dem Linearen Verfahren ist. Im Anschluß daran wird das Verfahren erörtert, das eingesetzt wurde, um eine Positionsmeldung des Lagemeßsystems auf eine Kante der Karte abzubilden (Map Matching). Schließlich wird die Implementierung der Klasse *DRMapMatching* vorgestellt. Eine Instanz dieser Klasse wird sowohl vom Client als auch von Server benötigt, um Soll-Positionen zu errechnen.

### 6.1 Datenstruktur für elektronische Karten

In diesem Abschnitt geht es zuerst einmal um die Entwicklung einer geeigneten Datenstruktur für elektronische Karten, die für die Anwendung des Algorithmus für kartenbasierte Koppelnavigation dienlich ist. Zunächst ist es dazu erforderlich, sich klarzumachen aus welchen Bestandteilen die Datenstruktur bestehen soll und welche Informationen diese enthalten müssen. Ausgehend davon werden die gewonnenen Erkenntnisse formalisiert. Im letzten Teilkapitel wird ein kurzer Überblick über bestehende Standards zur Repräsentation von elektronischen Karten gegeben.

#### 6.1.1 Bestandteile

Als Repräsentation für die Daten bietet sich eine Graphenstruktur an. Analog zu einer gedruckten Landkarte stellen die Knoten des Graphen Kreuzungen dar. Die Kanten entsprechen den Straßen bzw. Wegen zwischen den Kreuzungen. Um Bewegungsrichtungen auszudrücken, werden gerichtete Graphen benutzt.

Im Anschluß werden die Knoten näher betrachtet. Dann wird auf die Kanten eingegangen. Zu beiden Bestandteilen des Kartengraphen werden nähere Erläuterungen, insbesondere hinsichtlich der jeweiligen Attribute, gegeben.

##### 6.1.1.1 Knoten

Knoten stellen geographische Orte dar. Daher benötigen sie auf jeden Fall mindestens ein Attribut, das die Koordinaten speichert. Unsere Kartendaten, die wir von der Firma NavTech

erhalten haben, benutzen das Koordinatensystem WGS84, das von GPS verwendet wird. Mehr dazu später (im Kapitel 7).

Wie bereits erwähnt dienen Knoten dazu, Kreuzungspunkte zwischen mehreren Straßen zu modellieren. Da Kanten nur gerade Teilstücke einer Straße darstellen, ist es teilweise darüber hinaus erforderlich, Zwischenknoten (sogenannte „Shape-Points“) einzuführen, um Kurven approximieren zu können. Die geometrische Form einer Kurve wird demnach dadurch in der elektronischen Karte dargestellt, daß mehrere Zwischenknoten gesetzt werden. Diese werden durch Kanten verbunden und bilden so die Kurvenform nach. Der linke Teil der Abbildung 19 zeigt eine derartige Kurvenmodellierung!

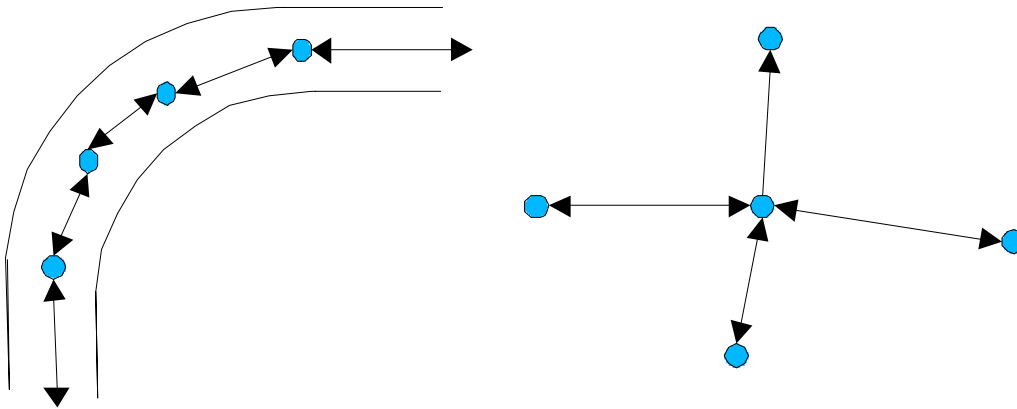


Abbildung 19 – Links: Modellierung einer Kurve durch Zwischenknoten („Shape Points“). Rechts: Knoten, die eine Kreuzung darstellen.

Während Kreuzungspunkte mehr als eine Ausgangskante besitzen, haben Shape Points lediglich eine Ausgangskante, was die Auswahl der vom Mobilien Objekt genommenen Kante eindeutig vorgibt! Deshalb sind Shape Points für den Dead-Reckoning-Algorithmus einfacher handzuhaben als Kreuzungen.

### Knotenattribute

An dieser Stelle sollen die Attribute zusammengestellt werden, die ein Knoten einer elektronischen Karte besitzen sollte. Nicht alle dargestellten Attribute werden für die Funktion des hier vorgestellten Kartenbasierten Dead-Reckoning-Algorithmus benötigt. Sie sind lediglich Ergänzungen, um den Informationsgehalt einer Karte zu erhöhen.

- Punktkoordinate (X,Y) oder (X,Y,Z) als WGS84-Koordinate
- bestimmter Typ des Knotens von Knoten (z.B. Parkplatz, ...)
- Bedingungsrelationen, die Regeln repräsentieren: Abbiegeverbot in Straße X wenn von Straße Y kommend, Geschwindigkeitsgebote, etc.
- beliebige Zusatzinformationen (z.B. Beschreibung eines Gebäudes für ein Touristeninformationssystem)

Zur Funktion des Map-Matchings ist im Grunde lediglich die Punktordinate des Knotens zwingend erforderlich. Die Knotenklasse und die Bedingungsrelationen können das Map Matching verbessern und erleichtern, stellen aber kein Muß dar.

### 6.1.1.2 Kanten

Eine Kante zwischen zwei Punkten A und B bedeutet, daß zwischen diesen beiden Orten ein Weg (Straße) existiert. Zum Einsatz kommen gerichtete Kanten. Dadurch ist es möglich, Einbahnstraßen oder getrennte Fahrwege für jede Richtung wie auf Autobahnen zu modellieren. Eine Straße, die in beide Richtungen befahrbar ist, wird folglich durch zwei Kanten repräsentiert: Für jede Richtung eine. (Die erste Kante von A nach B und eine zweite von B nach A.)

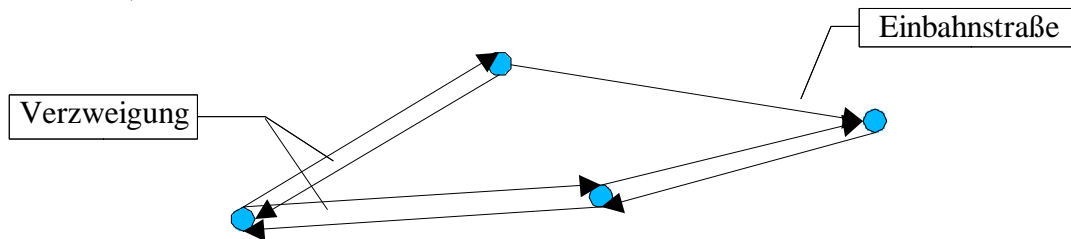


Abbildung 20 – Beispiel für den Einsatz von Kanten zur Darstellung von Verzweigungen und Einbahnstraßen.

Zur Optimierung könnte man mehrere Objektklassen für die Mobilien Objekte einführen (z.B. LKW, langsamer PKW, schneller PKW,...). Jeder dieser Objektklassen könnte man dann jeweils eine eigene Kante zwischen zwei Knoten zuweisen, um so verschiedene Fälle abdecken zu können.

Für das Kartenbasierte Verfahren mit Wahrscheinlichkeiten müßte man den einzelnen Kanten Wahrscheinlichkeitswerte zuweisen, damit das Verfahren funktionieren kann. Andere Kartenverfahren benötigen diese Zusatzdaten nicht.

Darüber hinaus kann man die Kanten mit einer Reihe von zusätzlichen Attributen ausstatten, je nachdem, für welchen Zweck die Karte zusätzlich genutzt wird. Bei einem Touristeninformationssystem oder bei einer Routenplanung wären Namen wichtig. Dadurch könnte sich der Benutzer informieren, daß er sich gerade in der „Breitwiesenstraße“ befindet und daß diese in seiner Bewegerichtung in die „Handwerkstraße“ mündet. Insbesondere bei einer Routenplanung wären noch die Anzahl der Fahrspuren von Interesse, um dem Anwender mitzuteilen wo er sich einordnen muß.

### Ausdrücken von Geschwindigkeitsveränderungen entlang einer Strecke

Statt die Momentangeschwindigkeit des Mobilien Objektes für die Berechnung der Soll-Positionen zu verwenden, könnte man den Kanten Geschwindigkeitsinformationen mitgeben. Diese könnten beispielsweise Durchschnittsgeschwindigkeiten sein. Auch Minimal-, Maximalgeschwindigkeiten oder nach sonstigen Kriterien festgelegte Geschwindigkeiten wären möglich. Die Informationen zur Geschwindigkeit der Kanten könnte man alternativ auch nur temporär einsetzen bis eine aussagefähige Momentangeschwindigkeit berechnet oder gemessen werden kann.

Um entlang einer Strecke zwischen zwei Punkten A und B einen Geschwindigkeitsverlauf darzustellen, ist es möglich, zwischen den beiden Endpunkten beliebig viele Zwischenpunkte einzuführen. Die Kanten zwischen den Zwischenpunkten können dann jeweils verschiedene Durchschnittsgeschwindigkeiten aufweisen!

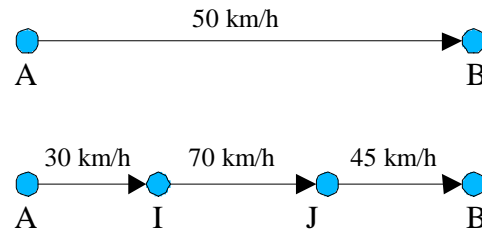


Abbildung 21 – Aufteilung einer Kante in mehrere Teilkanten, um verschiedene Geschwindigkeiten abzuspeichern.

### In Kanten Indirekt gespeicherte Informationen

Neben den oben beschriebenen expliziten Kantenattributen, besitzt eine Kante auch implizite Attribute. Das sind Attribute, die aus den expliziten Daten errechnet werden können. Daher ist es nicht nötig, diese ebenfalls zu speichern.

Indirekt speichern Kanten Informationen über Richtung  $\vec{r}$  und Entfernung  $e$  zwischen zwei Wegpunkten (Knoten). Man kann diese Attribute wie folgt aus den bekannten Daten errechnen:

$$e = \sqrt{(P1.x - P2.x)^2 + (P1.y - P2.y)^2} \quad \vec{r} = \begin{pmatrix} P2.x - P1.x \\ P2.y - P1.y \end{pmatrix}$$

$P1$  und  $P2$  sind dabei Start- bzw. Endknoten einer Kante.  $P.x$  bzw.  $P.y$  stellen die  $x$ - bzw. die  $y$ -Koordinate eines Punktes dar.

### Kantenattribute

Zusammenfassend sind für Kanten folgende Attribute denkbar:

- Geschwindigkeitsinformationen (Maximal-, Minimal-, Durchschnittsgeschwindigkeit, etc.)
- Objektklasse(n), für die die Kante gültig ist
- Wahrscheinlichkeit für Verkehrsfluß
- allgemeiner Name oder Bezeichner (z.B. „A81“, „Königstraße“)
- Anzahl der Fahrspuren

Wie bei den Knoten sind nicht alle Attribute für die Funktion des Kartenbasierten Algorithmus nötig. Genau genommen werden gar keine Attribute für Kanten vorausgesetzt. Das Verfahren, wie es hier vorgestellt wird, arbeitet gänzlich ohne Kantenattribute. Zum Teil könnten die Attribute aber für ein verbessertes Verfahren von Vorteil sein.

### 6.1.2 Formalisierung

Mathematisch betrachtet stellt die Datenstruktur für eine Karte einen gerichteten Graphen  $G = \{A_V, A_E\}$  dar. Die Knoten  $A_V$  und die Kanten  $A_E$  sind nichtleere Mengen, die wie folgt definiert sind:

**Knotenattributsmenge  $A_V$ :**  $A_V = \{ ((x,y,z), N, C_V, F_V) \}$  Menge von 6-Tupel mit

- $x, y, z$ : dreidimensionale Punktkoordinate
- $N$ : Bezeichner/Identifier des Objektes
- $C_V$ : Klasse/Typ des Knotens (Kreuzung, Parkplatz, ...)
- $F_V$ : Menge von weiteren Attributen des Knotens

**Kantenattributsmenge  $A_E$ :**  $A_E = \{ (s, e, N, G, P_{avg}, C_O, F_E) \}$  Menge von 7-Tupel mit

- $s \in A_V$ : Startknoten der Kante
- $e \in A_V$ : Endknoten der Kante
- $N$ : Bezeichner/Identifier der Kante
- $G$ : durchschnittliche/maximale Geschwindigkeit  $v_{avg}$  bzw.  $v_{max}$ , auf dieser Kante
- $P_{avg}$ : Wahrscheinlichkeit für Verkehrsfluß. Es gilt:  $\sum_{\forall \text{ ausgehende Kanten}} P_{avg} = 100\%$
- $C_O$ : Objektklasse für welche die jeweilige Kante gültig ist
- $F_E$ : Menge von weiteren Attributen der Kante

Ein Element  $x \in A_V$  stellt einen Knoten der Karte dar. Ein Element  $y \in A_E$  repräsentiert eine Kante der Karte, die an einem Knoten beginnt und zu einem anderen Knoten führt. Zu beachten ist, daß eine Kante jeweils einen eindeutigen Start- bzw. Endknoten besitzt, während ein Knoten mehrere eingehende und ausgehende Kanten haben kann.

### 6.1.3 Bestehende Standards

Die bisher getätigten Betrachtungen zur Datenstruktur der Kartendaten sind allgemeine Überlegungen. Ziel war es, eine Datenstruktur aufzustellen, die mindestens alle Anforderungen erfüllt, welche der kartenbasierte Algorithmus stellt. Ergänzend soll an dieser Stelle kurz auf existierende Standards für elektronische Kartendaten eingegangen werden.

Es existieren zur Zeit mehrere Standards von denen in Deutschland die geläufigsten GDF (Geographic Data File) und ATKIS (Amtlich Topographisch-Kartographisches Informationssystem) sind (aus [BER99]). In Amerika existiert der Standard SDTS (Spatial Data Transfer Standard). Während das Ziel von ATKIS darin besteht, Anwender mit einer Basis von raumbezogenen Daten zu versorgen, wurde GDF speziell für die Anwendung in der Fahrzeugnavigation entwickelt. Ein Unterschied zwischen GDF und ATKIS ist, daß GDF vor allem in den Kreuzungsbereichen eine detailliertere Erfassung besitzt. Wegen seiner Bedeutung für Fahrzeugnavigation wird nachfolgend etwas näher auf GDF eingegangen.

GDF ist ein europäischer Standard zur Beschreibung, Speicherung und zum Austausch von Straßennetzen und straßenbezogenen Daten. Neben der Spezifikation der Daten sind auch Objekte (sogenannte „Features“ im GDF-Datenmodell), Attribute und Relationen festgelegt.

Im Rahmen des Forschungsprogramms EUREKA der Europäischen Union wurde zur Standardisierung digitaler Straßenkarten für den Einsatz in Fahrzeugnavigationssystemen das Projekt DEMETER (Digital Electronic Mapping of European Territory) gegründet (siehe [CZO00]). In DEMETER wurden Spezifikationen für Dateninhalt, Datenerfassung und Austauschformat definiert. Daraus entwickelte sich der Entwurfsstandard GDF, der in der Version 1.0 im Oktober 1988 veröffentlicht wurde. Heute ist die Version 3.0 aktuell.

GDF speichert alle relevanten Informationen objektorientiert. Ein wichtiger Aspekt des

Standards ist das konzeptionelle Datenmodell, worin Objekte, Attribute und Relationen formell definiert werden.

Unter einem „Objekt“ versteht man einen vorhandenen oder geplanten Körper in der wirklichen Welt. Ein Objekt wird als „Feature“ bezeichnet. Features sind räumlich exakt begrenzt und zeitbeständig. Mobile Objekte, wie sie in dieser Arbeit vorkommen sind also keine „Features“ im Sinne von GDF! Jedes Feature kann als Punkt, Linie, Fläche oder als Kombination dieser sogenannten graphischen Primitiva dargestellt werden. Jedes Objekt gehört zu höchstens einer Objektklasse („Feature Class“).

Die Eigenschaften der Features werden durch Attribute wie Straßenart, Straßename, erlaubte Verkehrsrichtung, etc. beschrieben. Ein Objekt kann eine beliebige Anzahl an Attributen besitzen.

Beziehungen zwischen Features werden durch Relationen modelliert. Beispiele für solche Relationen sind: Abbiegeverbot von einer Straße in die andere, Vorfahrtsregelungen, etc. Auch Relationen können durch Attribute näher beschrieben werden.

Die Definition von Objekten, Attributen und Relationen findet sich in der GDF-Dokumentation [GDF95].

## **6.2 Map Matching**

Ein wesentlicher Bestandteil des kartenbasierten Algorithmus ist das Map Matching. Darunter ist der Abgleich einer vom Lagemeßgerät gelieferten Position mit den zur Verfügung stehenden Kartendaten zu verstehen. Der Grundgedanke ist, daß sich ein Mobiles Objekt nur auf den durch die Karte vorgegebenen Kanten befinden kann. Wegen der Ungenauigkeiten des Lagemeßsystems kann es dazu kommen, daß die gemeldete Position neben der Straße liegt. Daher wird die gemessene Lage auf die (anhand der vorliegenden Daten) bestimmte Karten-Kante projiziert.

In diesem Kapitel soll ein einfaches Map Matching Verfahren vorgestellt werden, das zur Auswahl einer passenden Kante der Karte führt, die zu einer Position am geeignetsten erscheint. Zunächst wird in die Problematik eingeführt, um besondere Schwierigkeiten beim Map Matching zu aufzuzeigen. Im Anschluß daran wird in mehreren Teilkapiteln das in dieser Arbeit eingesetzte Verfahren erläutert. Im letzten Teil werden dann weitere Map Matching Verfahren kurz vorgestellt.

### **6.2.1 Einführung in die Problematik**

Um in das Thema Map Matching einzuführen, werden zunächst ein paar grundlegende Situationen erörtert. Das soll dazu führen, daß dem Leser das Verständnis für die Problematik erleichtert wird. In den nachfolgenden Kapitel werden dann die hier gewonnenen Erkenntnisse algorithmisch gelöst und besprochen.

#### **6.2.1.1 Unkritischer Fall**

Das Objekt bewegt sich längs einer Kante der Karte, wobei bereits bekannt ist, welche Kante genommen wurde. Wie die Festlegung der Kante erfolgt ist, sei hier (vorerst) nicht von

Bedeutung.

Zum Abgleichen der gemessenen Ist-Position  $P$  mit der Karte genügt es, ein Lot auf die momentane Kante zu fällen. Der so bestimmte Lotfußpunkt  $P_L$  entspricht dann der gematchten Position.

Der Algorithmus für den Ablauf ist wie folgt:

1. Fülle das Lot auf die momentan benutzte Bewegungskante (die bereits bekannt ist!)
2. Die aktuelle Position des Objektes ist der Fußpunkt des Lotes auf der Kante zwischen zwei Knoten.

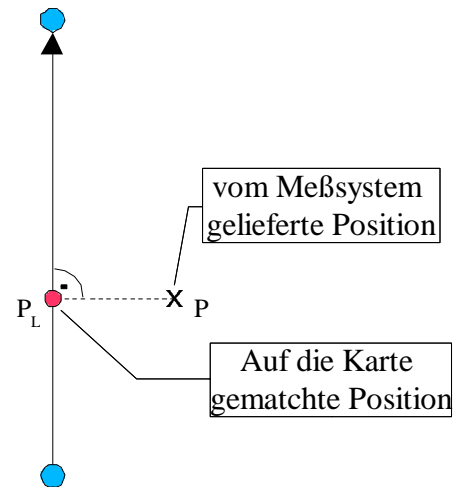


Abbildung 22 – Darstellung des einfachen Falls des Map Matching.

Die Lotfällung eines Meßpunktes auf eine ausgesuchte Kartenkante ist ein wesentliches Kennzeichen des Map-Matching-Algorithmus wie er in dieser Arbeit vorgeschlagen wird. (Ein Lagemeßpunkt wird immer durch suchen des Lotfußpunktes auf eine Kante gematcht.)

### 6.2.1.2 Kritische Fälle

Neben dem oben dargestellten unkritischen Fall sind auch zwei kritische Fälle zu beachten. Beide bestehen in der Nähe eines Knotens.

#### Nach Knotendurchlauf

Objekt hat gerade eben einen Knoten durchlaufen. Es ist aber noch nicht bekannt, welche abgehende Kante gewählt wurde.

Fest steht, daß alle vom soeben durchlaufenen Knoten abgehenden Kanten untersucht werden müssen, um eine korrekte Entscheidung für das Map Matching zu treffen.

Ein naiver Algorithmus wäre wie folgt (siehe Abbildung 23):

1. Bestimme kürzeste Entfernung zu allen vom Ausgangsknoten  $A$  ausgehenden Kanten (Lotrechte)
2. Die resultierende Kante, auf der ich mich gerade befinde ist somit gefunden (Kante mit der kürzesten Entfernung).
3. Aktuelle Position wird bestimmt durch das Lot der gemessenen Position auf die gefundene Kante.

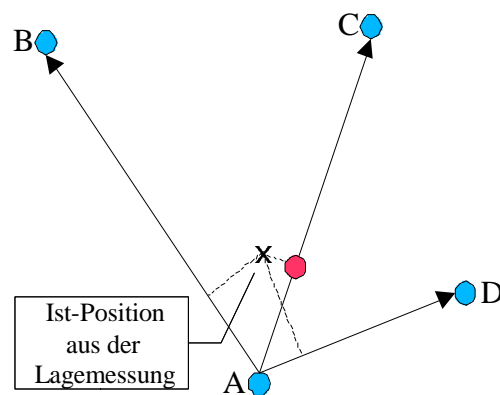


Abbildung 23 – Map Matching der Ist-Position zur Kante mit dem geringsten Abstand durch Fällung des Lotes.

Je nachdem wie häufig und wie genau das Positionierungssystem die Lage des Objektes

bestimmt, kann es anhand des oben beschriebenen naiven Algorithmus zu Fehlentscheidungen kommen! Wird die neue aktuelle Kante sehr kurz nach dem Durchlaufen des Knotens gesucht, ist die Wahrscheinlichkeit hoch, daß eine falsche Kante gewählt wird!

Später wird diese Fehlentscheidung bemerkt und zur einer anderen Kante gewechselt. Eventuell könnte sich der Wechsel mehrfach wiederholen bis spätestens zu dem Zeitpunkt bei dem die Entfernung der Kanten untereinander größer ist als die maximale Ungenauigkeit des Lagemesssystems.

Der größte Nachteil dieses Wechseleffektes ist, daß er impliziert, daß jedes Mal ein Lageupdate an den Server verschickt werden muß, wenn die Kante gewechselt wurde. Das widerspricht der Zielsetzung, ein Verfahren zu entwickeln, das mit möglichst wenig Updates auskommt!

Zur Lösung dieses Problems kann man wie folgt vorgehen: Man bestimmt die Orientierung der Bewegungsrichtung (entweder per elektronischem Kompaß oder durch Berechnung aus den letzten Positionen) und prüft neben der Entfernung zu einer Kante zusätzlich, ob der gefundene Richtungsvektor mit der Orientierung der gewählten Kante (annähernd) übereinstimmt. (Abweichungen sind wegen der Meßungenauigkeiten natürlich nicht auszuschließen!) Ein Beispiel bei der die Orientierung zum Map Matching verwendet wird ist in Abbildung 27 auf Seite 70 dargestellt.

Allein schon durch die Zuhilfenahme der Orientierung gewinnt man zusätzliche Sicherheit bei der Auswahl einer Kante. Aber um die Wahrscheinlichkeit für die richtige Kantenwahl zu erhöhen, ist es nötig, nicht sofort nach dem Knotendurchlauf nach einer neuen Kante zu suchen, sondern erst eine gewisse Entfernung  $m_{max}$  abzuwarten und dann nach einer neuen Kante zu suchen. Sinnvoller Weise sollte  $m_{max}$  so groß gewählt werden, daß es größer ist als der Fehler (d.h. maximale Abweichung) bei der Positionsbestimmung durch das Lagesystem.

Sobald eine neue Kante feststeht, ist das Problem behoben. Es geht weiter mit dem oben beschriebenen unkritischen Fall, bei dem eine Ist-Position lediglich durch Fällen des Lotes auf die Karte abgebildet wird.

### Vor Erreichen eines Knotens

In dieser Situation bewegt sich das Mobile Objekt auf einen Knoten zu, hat diesen aber (anhand der Meßwerte) noch nicht erreicht. Es stellt sich nun die Frage, ob es noch richtig ist, zur bisherigen Kante zu mappen oder eine Kante zu suchen. (siehe Abbildung 24)

Dieses Problem wird aber auch durch die Anwendung der oben vorgeschlagenen Lösung behoben. Das Mapping zur bisherigen Kante wird beibehalten, und es wird erst dann eine neue Kante gesucht, wenn die Ist-Position von der aktuellen Kante weiter entfernt ist als eine festgelegte maximale Entfernung  $m_{max}$ .

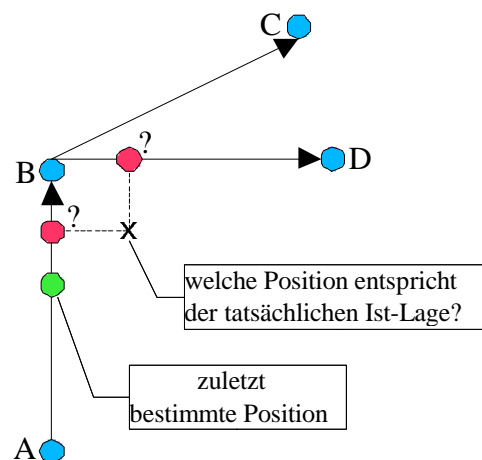


Abbildung 24 – Soll noch die alte Kante weiterbenutzt werden oder eine neue gesucht werden?

## 6.2.2 Geometrie einer Kante

Bevor es an die Beschreibung der Implementierung der Algorithmen beim Map Matching geht, ist es erforderlich, die Geometrie einer Kante näher zu betrachten. Dazu ist in Abbildung 25 eine einzelne Kante abgebildet. Die Punkte  $A$  und  $B$  markieren den Anfangs- bzw. Endknoten der Kante. Um die Kante ist in der Zeichnung ein Toleranzband gelegt. Dieses markiert die Punkte, die von der Kante die Entfernung  $m_{max}$  haben. Die Bezeichnung  $m_{max}$  ist zu unterscheiden von  $d_{max}$ . Während  $d_{max}$  der dem Algorithmus vorgegebene maximal tolerierbare Abstand zwischen Soll- und Ist-Position vorgibt, bezeichnet  $m_{max}$  die maximale Entfernung eines Punktes von einer Kante. Solange eine Position innerhalb des Toleranzbandes liegt, wird sie noch auf die aktuelle Kante gematcht. Ist eine Position weiter entfernt von der aktuellen Kante als  $m_{max}$ , dann muß eine neue Kante gesucht werden.

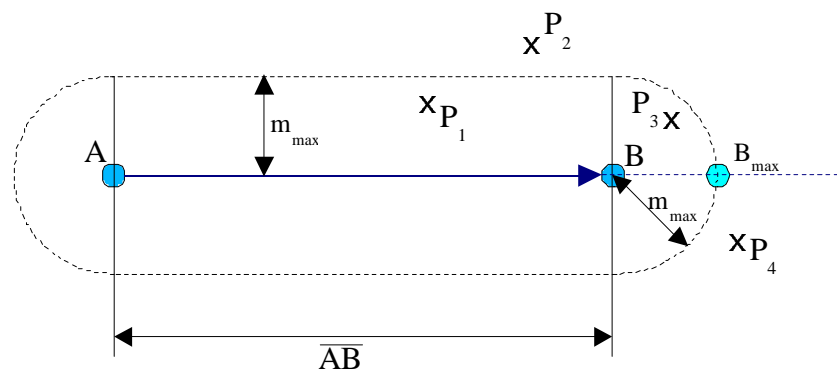


Abbildung 25 - Verdeutlichung der im Text benutzten Bezeichnungen für diverse Sachverhalte einer Kartenkante.

Neben der Geometrie soll noch eine Methode eingeführt werden, die später an mehreren Stellen im Map-Matching-Algorithmus Anwendung finden wird. Sie dient dazu, denjenigen Punkt der momentan verwendeten Kante zu bestimmen, der dem übergebenen Punkt  $P$  am nächsten ist. Die beiden Punkte  $P_1$  und  $P_2$  aus der Abbildung 25 liefern den jeweiligen Lotfußpunkt der Geraden durch  $A$  und  $B$  zurück. Dagegen haben die Punkte  $P_3$  und  $P_4$  den Punkt  $B$  als Rückgabewert. (Analog wäre es bei Punkte um  $A$ )

Der Abstand  $m_{max}$  bleibt dabei unberücksichtigt. Auch zu einem Punkt, der weiter von der Kante entfernt ist, wird der nächste Kantenpunkt zurückgeliefert.

```

findNearestLinkPoint( Kante K, Position P ) {
    P_perp = K.findPerpendicularPoint();

    if ( P_perp within line AB ) then {
        return P_perp;
    } else {
        d1 = distance( K.A, P );
        d2 = distance( K.B, P );
        if ( d1 < d2 ) then {
            return K.A;
        } else {
            return K.B;
        }
    }
}

```

```

    }
}

```

Der Punkt  $B_{max}$  ist ein besonderer Punkt, der für den Algorithmus von Bedeutung sein wird, wenn es darum geht, wann eine Nachfolgekante gewählt werden muß.  $B_{max}$  ist ein Punkt längs der Richtung der Kante von  $A$  nach  $B$ . Dabei hat  $B_{max}$  den Abstand  $m_{max}$  von  $B$ .

Die immer wieder auftauchenden Bezüge auf die Geometrie einer Kante, die in den Erläuterungen und in den wiedergegebenen Methoden der folgenden Kapitel vorkommen werden, beziehen sich stets auf die Nomenklatur, die durch Abbildung 25 festgelegt ist. Das gilt auch dann, wenn nicht explizit darauf hingewiesen wird.

### 6.2.3 Startposition

Das Verfahren benötigt einen definierten Startpunkt. Zwar reicht für die Funktion des Algorithmus bereits ein einziger Lagepunkt, der vom Lagemeßsystem geliefert wird. Aber um diese Anfangskoordinaten (bei Verwendung eines ungenauen Positionierungssystems) möglichst genau zu bekommen, sollte sich das Objekt über einen (von der Genauigkeit des Meßsystems abhängigen) hinreichend langen Zeitraum an einem Ort aufhalten, um mehrere Meßpunkte zu bekommen. Die so gesammelten Positionsinformationen werden einfach gemittelt und bilden einen Meßpunkt mit größerer Genauigkeit als ein einzelner Punkt. Dadurch wird die Wahrscheinlichkeit für ein falsches Mapping reduziert. Dieser gemessene Punkt dient dann als Anfangskoordinate für den Matching-Algorithmus.

Unabhängig von der Bestimmungsmethode des Startpunkts, muß man dann den so bestimmten Punkt mit der Karte abgleichen, um den momentanen Standort auf die Karte abzubilden. Das ist der aufwendigste Fall hinsichtlich der CPU-Zeit des Map-Matching-Algorithmus, weil es zu einem sehr komplexen Suchproblem führen kann.

In der Abbildung 26 ist das Problem verdeutlicht. Der vom Lagesystem ermittelte Punkt liegt am nächsten zu der Kante von  $X$  nach  $Y$ . (Die Wahrscheinlichkeit, daß sich das Objekt auf dieser Kante befindet ist daher sehr hoch.) Um aber feststellen zu können, welche Kante der Karte die minimale lotrechte Entfernung zur aktuellen Startposition besitzt, müssen aber alle Kanten der Karte betrachtet werden – wie man in der Abbildung erkennen kann. Das bedeutet einen ungeheueren Rechen- bzw. Zeitaufwand, da die Anzahl der Punkte einer realen Karte sehr hoch ist. Würde man nur die nähere Umgebung absuchen, käme man zu dem (voreiligen) Schluß, daß sich das Objekt auf der Kante  $AC$  befindet. Erst die globale Betrachtung aller Kanten der Karte kann eine verlässliche (und korrekte) Aussage liefern.

Anschaulich gesprochen, müßte man bei der Initialisierung auch eine Kante betrachten, die von Kempten nach Flensburg

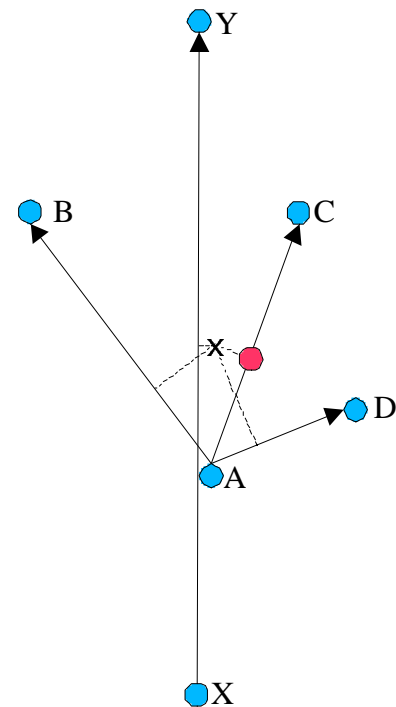


Abbildung 26 - Verdeutlichung des Initialisierungsproblems. Wenn nicht sämtliche Kartendaten betrachtet werden, kann es zu einer falschen Zuordnung kommen (roter Punkt).

führt (wenn es beispielsweise eine derartige Autobahn gäbe, die zudem kurvenfrei wäre)! Die Suche nach Kanten bei der Initialisierung wäre keineswegs lokal auf einen überschaubaren Umkreis beschränkt! Der maximale Suchradius, den man bei der Initialisierung zu betrachten hat, wird durch die längste Kante der gegebenen Kartendaten beschränkt. Ist also bekannt, daß beispielsweise die längste Kante zwischen zwei Punkten 15km lang ist, dann kann man sich darauf beschränken, nur alle Knoten im Umkreis von 15km von der aktuellen (Start-)Position zur Positionsbestimmung zu betrachten. Ist die Länge der längsten Kante dagegen unbekannt, muß man die ganze Karte durchforsten.

Man kann den Suchradius künstlich noch reduzieren, indem man die Kartendaten so aufbaut, daß man zusichert, daß beispielsweise alle 2km ein Knoten folgen muß, auch wenn das gerade Teilstück einer Straße länger als diese Schranke ist. Man führt also Zwischenpunkte ein, um den Suchradius zu reduzieren.

Eine zusätzliche Vereinfachung ließe sich dadurch realisieren, wenn man die Datenbasis der Kartendaten auch nach Orientierungen durchsuchen könnte. Zusätzlich müßte das Meßsystem in der Lage sein, Orientierungsdaten bereitzustellen (z.B. elektronischer Kompaß). Dann wäre es möglich eine Anfrage wie die folgende zu formulieren: „Wähle alle Kanten im Umkreis von  $x$  km um den Punkt  $P$ , welche die Orientierung  $\alpha \pm 30^\circ$  haben“. Durch zusätzliche Angabe die Orientierung wäre die Menge der Kanten, die für das Map-Matching untersucht werden müssen, bereits durch die Datenbank reduziert.

Da anzunehmen ist, daß die Datenbank über Indizes und geschickte Datenhaltung eine komplexe Anfrage in deutlich geringerer Rechenzeit durchführen kann, ist das effizienter als sich alle Kanten im Umkreis von  $x$  zum Punkt  $P$  von ihr geben zu lassen und erst anschließend die jeweiligen Vergleiche selbst (in einer sequentiellen Suche) durchzuführen.

Der Algorithmus zur Neuinitialisierung unter der Annahme, daß  $x$  km die größte Länge einer Kante ist, sieht wie folgt aus. Außerdem sei nur eine Suche auf Knotenbasis ohne Orientierungsangabe möglich.

```

initialize ( Position currentP ) {
    ResultLink = null;
    ResultDistance =  $\infty$ ;

    Nset = "Find all nodes within distance x from currentP";

    for ( all nodes N' in Nset ) {
        for ( all outgoing links Lout of N' ) {
            tempDistance
                = distance( findNearestLinkPoint( Lout,
                                                    currentP ),
                            currentP );

            if ( tempDistance < ResultDistance ) then {
                ResultDistance = tempDistance;
                ResultLink = Lout;
            }
        }
    }
    return ResultLink;
}

```

---

```

}
```

Der oben dargestellte Algorithmus zur Initialisierung sucht stets die Kante aus, die am nächsten zur gemessenen Position ist. Sofern das Mobile Objekt über keine Möglichkeit verfügt, zu diesem Zeitpunkt einen Richtungsvektor zu messen, ist auch nichts anderes möglich. Dieser Ansatz birgt aber einen gravierenden Nachteil: Es ist beispielsweise nicht feststellbar, ob sich das Objekt auf der gerichteten Kante von A nach B oder von B nach A befindet. Da die Kanten zwischen den Knoten gerichtet sind, macht dies durchaus einen Unterschied!

Um das Map-Matching mit größerer Wahrscheinlichkeit auf die richtige Kante durchzuführen, muß aber neben der Position auch die Bewegungsrichtung berücksichtigt werden. Das ist auch bei Lagemeßsystemen möglich, die keinen elektronischen Kompaß besitzen. Hier kann aus den letzten gemessenen Positionen eine Bewegungsrichtung errechnet werden. Dieser Bewegungsvektor kann bereits ab 2 vorhandenen Meßpunkten ermittelt werden. Mit steigender Anzahl der herangezogenen Punkte, steigt auch die Genauigkeit mit der die Richtung errechnet werden kann, weil durch die steigende Anzahl der Punkte einzelne Positionsmeßfehler das Ergebnis weniger stark beeinflussen.

Dabei gilt es aber abzuwägen, welche Punktzahl die besten Ergebnisse liefert. Der Grund ist, daß der durch Mittelung bestimmte Richtungsvektor immer träger auf Richtungsänderungen (Kurvenfahren oder Richtungsumkehr) reagiert, je mehr zurückliegende Punkte berücksichtigt werden. (siehe Kapitel 5.1)

Abschließend soll der Initialisierungsalgorithmus vorgestellt werden, der zum Zuge kommt, wenn neben einer Position zusätzlich eine Orientierung zur Verfügung steht:

```

initialize_2 ( Position currentP, Orientation alpha ) {
    ResultLink = null;
    ResultDistance = ∞;

    Nset = "Find all nodes with distance x from currentP";

    for ( all Nodes N' in Nset ) {
        for ( all outgoing links Lout of N' ) {
            tempDistance
                = distance( findNearestLinkPoint( Lout,
                                                    currentP ),
                            currentP );

            if ( tempDistance < ResultDistance ) then {
                if ( |Lout.orientation-alpha| < εOrientation ) {
                    ResultDistance = tempDistance;
                    ResultLink = Lout;
                }
            }
        }
    }
    return ResultLink;
}

```

Der einzige Unterschied zwischen dem ursprünglichen Algorithmus und dem neuen, der die

Orientierung berücksichtigt, ist die zusätzliche *if*-Abfrage, die nur diejenigen Kanten auswählt, deren Orientierung sich um weniger als einem Schwellwert  $\epsilon_{\text{Orientation}}$  von der momentanen Orientierung unterscheiden.

Bei Lagemesssystem, die keine Orientierung bestimmen können, bleibt zur Initialisierung nichts anderes übrig, als zunächst die Initialisierung nach der Methode *initialize* durchzuführen. Sobald sich das dadurch getroffene Matching als falsch herausstellt oder sobald aus mindestens 2 Positionsmessungen eine Bewegungsrichtung errechnet werden kann, ist es möglich *initialize\_2* aufzurufen.

### **Verändert die Verwendung eines elektronischen Kompaß die Performance des Verfahrens?**

Besitzt das verwendete Lagesystem die Möglichkeit, neben der momentanen Position auch im Stand eine Richtung zu bestimmen, ist es anhand dieser Richtung schon bei der Initialisierung möglich, herauszufinden in welche Richtung sich das Mobile Objekt bewegt bzw. bewegen wird. Für den Algorithmus bedeutet das, daß schon bei der Initialisierung zu richtigen Kante gemappt wird.

Besitzt das Meßsystem dagegen keinen elektronischen Kompaß, ist es bei der Initialisierung nicht möglich, zu bestimmen, ob ich mich auf der Kante von A nach B in Richtung B oder in Richtung A bewegen werde. Die Entscheidung kann erst fallen, wenn sich das Objekt in Bewegung gesetzt hat. Folglich benötigt man bei einem solchen Fall zwei Updates an den Server. Eine Nachricht muß nach der Initialisierung erfolgen, in der dem Server die aktuelle Position mitgeteilt wird. Dabei besteht die Unsicherheit, die falsche Kante ausgesucht zu haben, weil keine Orientierung zur Verfügung steht. Eine weitere Nachricht ist später erforderlich, um entweder die Position auf Grund des korrigierten Map Matchings zu melden, oder um nun die Bewegungsrichtung mitzuteilen, weil sich das Objekt in Bewegung gesetzt hat. Erst jetzt kann das System zuverlässig auf die Karte matchen, weil die Kante durch die Orientierung eindeutig bestimmt werden kann.

Im Falle des Meßsystems mit Kompaß kann man zwar bereits bei der Initialisierung das Matching mit der Karte durchführen, weil neben der Position auch die (zukünftige) Bewegungsrichtung bekannt ist. Man muß aber trotzdem ein Update verschicken, sobald sich das Objekt in Bewegung gesetzt hat, da zum Zeitpunkt der Initialisierung nicht bekannt ist, wie lange das Objekt noch an diesem Punkt verharret.

Folglich ist es von der Anzahl der gesendeten Nachrichten her betrachtet unerheblich, ob das Lagesystem über einen Kompaß verfügt oder nicht.

### **6.2.4 Kantenübergang**

Wenn sich das Mobile Objekt bewegt, was der im Betrieb zu erwartende Normalfall sein wird, wird früher oder später eine Kartenkante verlassen und zur nächsten übergewechselt. Folglich ist es die Aufgabe des Algorithmus, einen solchen Fall zu erkennen und eine neue Kante auszuwählen. Ein weiteres Problem ist, daß es vorkommen kann, daß eine vom Algorithmus ausgesuchte Matching-Kante sich als falsch herausstellt. Anders formuliert geht es darum, was passieren soll, wenn das Matching nicht stimmt.

Darüber hinaus ist in diesem Zusammenhang auch die Frage zu klären, wie ein solcher Fall erkannt werden kann. Es soll schließlich verhindert werden, daß es zu (falschen) Lageupdates kommt nur weil das Objekt ständig auf einer anderen Strecke vermutet wird!

#### 6.2.4.1 Erkennung von falschem Matching

Die Methode „correctCurrPos“ ist der eigentliche Startpunkt der Bearbeitung. Sie wird nachfolgend in Pseudocode wiedergegeben. Angestoßen wird diese Methode vom Hauptprogramm (siehe Kapitel 4.4, Seite 36) und ist Teil der Klasse *DRMapMatching*. Zunächst wird die momentane Lage (Ist-Position),  $P_M$ , durch das Meßsystem bestimmt und der Methode als Parameter übergeben. Dieser Punkt wird auf die aktuelle Kante gemappt: Punkt  $P_L$ . Anschließend wird verglichen, ob die bisher benutzte Kante nicht bereits verlassen wurde. Dies ist der Fall, wenn der Meßpunkt vom nächstgelegenen Kantenpunkt weiter als ein vorgegebener Schwellwert  $m_{max}$  entfernt ist. Sollte das eingetreten sein, wurde die momentane Kante verlassen – das Matching stimmt also nicht mehr. Es muß eine neue Kante gefunden werden (*findNewLink*). Schließlich wird  $P_L$  zurückgeliefert.

```

correctCurrPos( Position  $P_M$  ) {
     $P_L$  = findNearestLinkPoint( currentLink,  $P_M$  );

    if (  $|P_M - P_L| > m_{max}$  ) then {
        newLink = findNewLink( currentLink,  $P_M$  );
         $P_L$  = findNearestLinkPoint( newLink,  $P_M$  );
        currentLink = newLink;
    }

    return  $P_L$ ;
}

```

#### 6.2.4.2 Finden einer neuen Kante

Voraussetzung zur Anwendung einer Matching-Strategie ist, daß erkannt wurde, daß die Kante auf die bisher gematcht wurde nicht mehr paßt. Hierzu sind zwei Fälle zu unterscheiden.

1. Das Objekt hat sich lediglich von der aktuellen Kante wegbewegt. (Es wurde eine Abzweigung genommen oder entlang einer Kurve bewegt)  
In jedem Fall wurde ein Knoten passiert.
2. Bisheriges Mapping war bereits falsch. (Bei der Initialisierung wurde zur falschen Kante gemappt oder es wurde nach Durchlauf eines Knotens auf die falsche Ausgangskante weitergemappt)

Je nach aufgetretenem Fall muß anders reagiert werden. Im 1. Fall wird „Forwardtracking“ benutzt, während der 2. Fall durch die Strategie des „Backtracking“ gelöst wird. Beide Algorithmen werden später näher vorgestellt.

Nachfolgend wird der Algorithmus zur Entscheidung, welche Strategie verfolgt werden soll, im Pseudocode wiedergegeben.

```

findNewLink( Link L, Position P ) {
     $P_L$  = findNearestLinkPoint( L, P );
}

```

```

    if ( PL == L.B ) then {
        newLink = forwardtracking( L.B, P );
    } else {
        newLink = backtracking( L, P );
    }

    return newLink;
}

```

Aufgerufen wird `findNewLink` durch die im Kapitel 6.2.4.1 vorgestellte Methode `correctCurrPos`. Das geschieht immer dann, wenn der Abstand zwischen der gemessenen Lage und dem kantennächsten Punkt größer als  $m_{max}$  wird. Dadurch wird sowohl der Fall des falschen Mappings als auch das Problem des Übergangs von einer Kante zur nächsten abgedeckt.

### Forwardtracking

Beim Forwardtracking geht es darum, ausgehend vom Endpunkt der aktuellen Kante die Kante zu bestimmen, welche als nächstes vom Objekt verwendet wird. Dazu sind alle vom Endpunkt ausgehenden Kanten zu betrachten.

Diejenige Kante wird ausgewählt, zu welcher der Fußpunkt des Lotes von der aktuellen Position am nächsten ist und gleichzeitig der Abstand keiner ist als  $d_{max}$ . Existiert für alle Kanten kein Lotpunkt, der diese beiden Bedingungen erfüllen kann, ist davon auszugehen, daß das Matching (der alten Kante) fehlerhaft war. Es muß eine komplette Neuinitialisierung erfolgen.

```

forwardtracking( Node startNode, Position currentP ) {
    ResultLink = null;
    ResultDistance = ∞;
    for ( all outgoing links outgoingLink of startNode ) {
        tempDistance
            = Distance( findNearestLinkPoint( outgoingLink,
                                                currentP ),
                       currentP );

        if ( tempDistance < ResultDistance ) then {
            ResultDistance = tempDistance;
            ResultLink = outgoingLink;
        }
    }

    if ( ResultDistance > mmax ) then {
        ResultLink = initialize( currentP );
    }

    return ResultLink;
}

```

*Anmerkung:* Man könnte den Algorithmus zum Forwardtracking analog zu `initialize_2` auch durch kleine Änderungen so umbauen, daß neben dem Abstand zwischen Meßpunkt und

Lotfußpunkt noch zusätzlich eine Orientierung verwendet wird. Dadurch ist eine Verbesserung der Matching-Eigenschaften zu erwarten! Das Objekt muß sich schließlich nicht immer auf der Kante befinden, die am nächsten zum Meßpunkt liegt! Insbesondere sehr kurz nach dem Durchlaufen eines Knotens liegen alle Ausgangskanten sehr dicht beieinander! Schon ein kleiner Fehler bei der Positionsbestimmung führt zur Wahl einer falschen Kante.

Durch die zusätzliche Verwendung der Orientierungsinformationen ließe sich die Auswahlssicherheit verbessern.

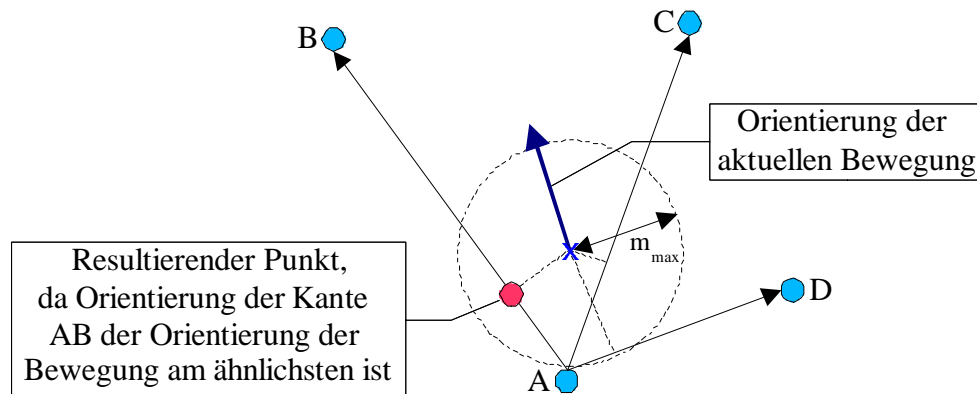


Abbildung 27 – Beispiel, bei dem das Forwardtracking ohne Berücksichtigung der Orientierung ein anderes Resultat liefern würde. Ohne Orientierung wäre das Resultat ein Punkt auf der Kante AC. Mit Berücksichtigung der Orientierung wird die (weiter entfernte) Kante AB ausgesucht.

## Backtracking

Das Backtracking ist erforderlich, wenn die momentan gewählte Kante nicht richtig ist. Das liegt daran, daß nach dem letzten Knotendurchlauf – aus welchen Gründen auch immer – die falsche Kante gewählt wurde. Sollte es seit der Neuinitialisierung keinen Knotendurchlauf gegeben haben, muß unweigerlich erneut initialisiert werden, da keine Möglichkeit des Zurücksetzens gegeben ist.

Das Backtracking muß dann angestoßen werden, wenn festgestellt wurde, daß der Abstand zur momentanen Kante zwar größer ist als  $m_{max}$ , aber der Lotfußpunkt noch zwischen dem Startpunkt A und dem Endpunkt B der momentanen Kante liegt.

```

backtracking ( Link currLink, Position currentP ) {
    A = currLink.A; // A = current links's starting node

    if ( A already visited ) then {
        ResultLink = forwardtracking ( A, currentP );
    } else {
        ResultLink = initialize( currentP );
    }

    return ResultLink;
}

```

Anmerkung: Um die Abfrage zu ermöglichen, ob „A bereits besucht“ wurde, ist es nötig, sich

in einer Variablen stets zu merken, welcher Knoten zuletzt besucht wurde. Das wäre kein großer zusätzlicher Speicheraufwand, aber alle hier wiedergegebenen Algorithmen müßten so ergänzt werden, daß wenn eine neue Kante über das Forwardtracking ausgewählt wurde, diese Hilfsvariable entsprechend gesetzt wird. Nach einer Neuinitialisierung wäre diese Variable auf „null“.

Alternativ könnte man beim Backtracking generell Forwardtracking auf dem Kantenanfangspunkt A durchführen. (Also die if-Abfrage ersatzlos streichen) Dadurch würde Forwardtracking zwar zunächst alle anderen Ausgangskanten von A betrachten, aber trotzdem bei Nichterfolg eine Neuinitialisierung anstoßen. Das Resultat wäre das selbe, nur daß ein paar zusätzliche Rechenschritte benötigt werden.

### 6.2.5 Verlassen des Kartenbereichs

In realen Anwendungen kann es passieren, daß der Algorithmus zur Kartenverfolgung nicht weiterbenutzt werden kann. Gründe dafür können sein, daß die Kartendaten nicht hinreichend detailliert sind oder daß der von der Karte abgedeckte Bereich verlassen wird. Ein Beispiel für die fehlende Detaillierung könnte sein, daß die Kartendaten kleinere Seitenstraßen oder Gassen nicht enthalten (siehe dazu auch Kapitel 3.2.2, Seite 23). Beim Verlassen des Kartenbereichs ist es beispielsweise vorstellbar, daß Kartendaten nur den Großraum Stuttgart umfassen. Fährt ein Fahrzeug aber auf der Autobahn nach München, wird der Bereich zu einem bestimmten Zeitpunkt zwangsläufig verlassen, und die Kartendaten sind nicht mehr von Nutzen.

Das Koppelnavigationsprotokoll muß daher in der Lage sein zu erkennen, sobald es nicht mehr möglich ist, Map-Matching zu betreiben. In einem solchen Fall muß auf einen alternativen Algorithmus übergegangen werden. Hierbei bietet sich der Algorithmus für das lineare Fortsetzen an, der immer funktioniert, weil er ohne Zusatzinformationen auskommt.

#### Möglichkeiten zur Erkennung

Zum Map-Matching bzw. Dead-Reckoning werden in den vorgestellten Verfahren stets zwei Kenngrößen herangezogen: Abstand zu einer Kante und Bewegungsrichtung des Mobilien Objektes. Anhand dieser beiden Parameter ist es auch möglich, zu erkennen, wann die Karte verlassen wurde.

Ist a priori bekannt, welchen Bereich die Kartendaten abdecken, genügt es zu prüfen, wann dieser Bereich verlassen wird. Dazu ist es bereits ausreichend, zwei Punktkoordinaten (linker oberer Punkt, rechter unterer Punkt) des Begrenzungsrechtecks einer Karte zu speichern. Für eine flexiblere Eingrenzung des Gebietes könnte man statt eines Rechtecks auch ein Polygonzug oder eine andere geometrische Form zulassen. Sobald festgestellt wird, daß sich das Objekt außerhalb dieser Begrenzung befindet, muß zwangsläufig auf einen Ausweichalgorithmus umgeschaltet werden.

Umgekehrt ist es zur Minimierung der Anzahl an benötigten Updatenachrichten nötig, immer wieder zu prüfen, ob zum Map-Matching-Verfahren zurückgekehrt werden kann. Das ist dann möglich, wenn der von der Karte abgedeckte Bereich erneut betreten wird. (Außerhalb dieses Bereiches ist es nicht nötig, ein Map-Matching zu probieren.) Wird dies festgestellt, muß die aktuelle Positionsmessung mit der Karte abgeglichen werden (Initialisierung) und das

normale Map-Matching-Verfahren kann wieder ablaufen. Diese Prüfung muß der Client durchführen, weil nur ihm die Ist-Position bekannt ist, die zur Entscheidung benötigt wird, ob sich das Mobile Objekt außerhalb oder innerhalb des Kartenbereichs befindet.

Bisher wurde angenommen, daß der Bereich für den die Karte gültig ist, bekannt ist. Im folgenden soll geschildert werden, wie das Verfahren ablaufen kann, wenn dies nicht der Fall ist. Ein Verlassen der Karte ist dann dadurch feststellbar, daß das Koppelnavigationsprotokoll keine Kartenkante finden kann, die weniger als die maximale Ungenauigkeit des Lagemeßsystems von der aktuellen Position entfernt ist.

Für den umgekehrten Fall der Prüfung, ob wieder Map-Matching betrieben werden kann, ist der Rechenaufwand dagegen deutlich höher als bei der Bereichsprüfung. Wie bei einer Initialisierung des Map-Matchings (siehe dort) müssen im schlechtesten Fall alle Kanten der Karte durchlaufen werden, um die Kante zu finden, die der aktuellen Position am besten entspricht. Das bedeutet viel mehr Aufwand als nur zu prüfen, ob die momentane Position innerhalb oder außerhalb eines definierten Bereiches ist.

Daher ist es nicht sinnvoll, den Algorithmus zu benutzen, wenn der von der Karte abgedeckte Bereich nicht explizit bekannt ist, weil immer wieder sehr viel Rechenzeit bei der Prüfung, ob die Karte wieder betreten wurde, unnötig verschwendet würde. Um diesen Fall zu entgehen, kann man statisch für eine Karte den Geltungsbereich bestimmen. Diese Bestimmung ist nur ein einziges Mal nötig, wenn man das Ergebnis dauerhaft mit den Kartendaten speichert. Anschließend kann die Prüfung, ob sich das Objekt innerhalb oder außerhalb des Kartenbereiches befindet, anhand dieses Ergebnisses erfolgen.

Für beide Fälle (mit oder ohne explizit bekannten Kartenbereich) gilt, daß die Häufigkeit der Prüfung, ob der Kartenbereich erneut betreten wurde, davon abhängt, wie effizient, diese Prüfung durchgeführt werden kann. Ist es problemlos möglich, zwischen zwei neuen Positionsmeldungen des Lagemeßsystems, zu entscheiden, ob sich das Objekt wieder im Kartenbereich befindet, ist das sicherlich die optimale Lösung, weil sofort auf den Wiedereintritt in den Datenbereich reagiert werden kann. Dauert diese Prüfung dagegen länger, kann sie nur in längeren Zeitabständen durchgeführt werden. Ein weiterer Grund, die Prüfung nicht nach jeder Positionsbestimmung durchzuführen, ist den Stromverbrauch beim Mobilien Objekt mit Akkubetrieb zu verringern. Jede Berechnung verbraucht zusätzlichen Strom und verringert folglich die Betriebszeit.

Eine Überprüfung, ob der Kartenbereich wieder betreten wird, ist nicht von Server durchführbar, da ihm die Ist-Positionen fehlen, um eine Entscheidung zu treffen. Das heißt, daß der Client allein dafür verantwortlich ist!

### **Übergang zu einem anderen Algorithmus**

Nachdem erkannt wurde, daß der gegenwärtige Algorithmus nicht mehr adäquat ist, muß auf einen anderen Algorithmus übergegangen werden. Da sowohl das Mobile Objekt als auch der Lokationsserver immer den selben Algorithmus benutzen müssen, ist es vor dem Übergang erforderlich, zwischen den beiden einen Folgealgorithmus auszuwählen.

Hierzu ist es entweder nötig, daß Client und Server jedes Mal bei Bedarf über einen Folgealgorithmus „verhandeln“ oder daß sich die beiden einmalig auf einen „Default-Fallback-Algorithmus“ einigen, auf den immer automatisch übergegangen wird, wenn es nötig ist.

Als Fallback-Algorithmus bietet sich immer der Algorithmus des Linearen Fortsetzens an, da er auf keinerlei Zusatzdaten zurückgreift und somit immer anwendbar ist. Es handelt sich dabei um den einfachsten Dead-Reckoning-Algorithmus. Aber natürlich ist es möglich, sich auf einen anderen Algorithmus einzulassen.

Für die Rückkehr vom Fallback-Algorithmus auf den kartenbasierten Ansatz, muß dann lediglich eine Nachricht ausgetauscht werden, in der dem Kommunikationspartner mitgeteilt wird, daß ab sofort wieder Map-Matching verwendet wird.

Bereits im Kapitel 4.6 (ab Seite 39) wurde ein Kommunikationsprotokoll zwischen Client und Server präsentiert, in dem die Möglichkeit gegeben ist, auf ein anderes Koppelnavigationsverfahren zu wechseln. Dort wird kein Fallback-Algorithmus beim Verbindungsaufbau vereinbart, sondern ein Folgealgorithmus bei Bedarf zwischen den beiden Parteien ausgehandelt.

### 6.2.6 Weitere Map Matching Verfahren

Neben dem für diese Arbeit entwickelten und in diesem Kapitel vorgestellten Map Matching Verfahren, existieren auch diverse andere Verfahren. Im folgenden werden ein paar weitere Verfahren präsentiert. Es ist zu beachten, daß es sich nicht um Verfahren für Koppelnavigation eines Location Based Service handelt – wie sie in dieser Arbeit besprochen werden, sondern lediglich um Verfahren zum Kartenabgleich. Die dadurch ermittelten Positionsinformationen bleiben lokal im Fahrzeug und werden nicht an andere Übertragen! (Nachfolgende Beschreibungen basieren auf [RAB99].)

Das wohl älteste Verfahren ist das *Halbdeterministische Map Matching*. Mit diesem Verfahren begann Anfang der 70er Jahre die Entwicklung von Map Matching Verfahren. Kennzeichnend ist, daß das Navigationssystem mit einer Anfangsposition manuell initialisiert werden muß. Das Fahrzeug darf zudem auf keinen Fall von den Straßen der digitalen Kartendatenbank abweichen. Wird der Kartenbereich verlassen, bricht das Verfahren ab. Eine automatische Reinitialisierung ist nicht möglich, weil wieder ein manueller Startpunkt vorgegeben werden muß. Daher stammt die Bezeichnung halbdeterministisch.

Das Verfahren benötigt keine absoluten Sensoren (wie GPS), sondern verwendet üblicher Weise Odometer und Gyroskop oder differentielle Odometer mittels Koppelnavigation. Da diese Sensoren statistische Fehler aufweisen, die sich mit der Zeit summieren, werden die Sensoren nach jedem Abbiegevorgang neu kalibriert. Dazu muß aber ein Abbiegevorgang erkannt werden!

Das halbdeterministische Verfahren wurde zum *Wahrscheinlichkeitsbasierten Map Matching* weiterentwickelt. Der große Vorteil des neuen Verfahrens bestand darin, daß eine automatische Reinitialisierung möglich ist, wenn der Kartenbereich verlassen wurde und später wieder betreten wird. Das Verfahren beruht darauf, daß die Sensorfehler des Positionierungsmoduls statistisch gut erfaßt werden. Die Sensorfehler werden mittels Fehlerfortpflanzung zur Berechnung der statistischen Fehler der Positionsinformationen aus diesem Modul herangezogen. Mit Hilfe geeigneter statistischer Modelle und Verfahren läßt sich daraus ein Vertrauensintervall bilden, mit welchem eine erste Auswahl der Elemente aus der Kartendatenbank erfolgt. In der Regel wird das Vertrauensintervall als zweidimensionale Ellipse angegeben.

Alle Elemente der Kartendatenbank werden darauf hin extrahiert, die im Vertrauensintervall liegen. Die Elemente dieser Menge werden dann weiteren Prüfungen unterzogen. Es werden die Orientierung, die Entfernung zur momentanen Position, die Verknüpfung mit einem zuletzt durchlaufenen Element und weitere Restriktionen geprüft. Das Ergebnis ist, daß ein passendes Element ausgesucht wird, auf dem sich das Fahrzeug befindet.

Ein System, das 1985 in den USA auf den Markt kam, ist ETAK Navigator. Es war das erste kommerziell erhältliche unabhängige Fahrzeugnavigationssystem, das auf Map Matching basiert. Das System benutzte lediglich ein differentielles Odometer und einen elektronischen Kompaß.

Während die bisherigen Verfahren die Elemente aus der elektronischen Karte mittels einer klaren Abgrenzung auswählen, bei dem ein Segment entweder eine Bedingung erfüllt oder nicht, geht das *Fuzzy-Logik-basierte Map Matching* anders vor. Durch die Fuzzy-Logic werden Entscheidungen nicht mittels der beiden Zustände *ja* und *nein* der gebräuchlichen zweiwertigen Logik getroffen, sondern über einen kontinuierlichen Bereich von 0 bis 1 der Wahrheitswerte. Analog zur zweiwertigen Logik existieren Operatoren wie UND und ODER. Zusätzlich werden WENN-DANN-Regeln benutzt, um Bedingungen auszudrücken.

Das Fuzzy-Logic Verfahren soll vorallem dann zu besseren Map Matching Resultaten gegenüber den Verfahren mit zweiwertiger Logik führen, wenn viele Elemente der Karte dicht beieinander liegen (z.B. in der Innenstadt).

Ein Fuzzy-Logic-Verfahren wurde zu Beginn der 90er Jahre entwickelt und 1994 zum Patent angemeldet [KAO94].

Ein weiteres Verfahren ist *Kalman-Filter-basiertes Map Matching*, bei dem ein Kalman-Filter eine Schätzung des aktuellen Zustands oder auch eine Vorhersage des zukünftigen Zustands eines diskreten, stochastisch gestörten, linearen Systems liefert.

Beim *Krümmungsbasierten Map Matching* (siehe [BER99]) geht man von einem gänzlich anderen Ansatz aus. Hier werden Krümmungen, welche im Positionierungs-Modul gemessen werden, mit den Krümmungen der digitalen Kartendatenbank verglichen. Anhand übereinstimmender Krümmungen wird das Map Matching durchgeführt.

## 6.3 Ablauf des Verfahrens

Wie bereits beim Linearen Fortsetzen soll auch beim Kartenbasierten Algorithmus der Ablauf des Verfahrens anhand des Pseudocodes für die Klasse erläutert werden. An dieser Stelle wird der Hinweis wiederholt, daß identische Methoden der Klasse sowohl beim Client als auch beim Server in den Programmablauf eingegliedert sind. Informationen dazu sind im Kapitel Kapitel 4 (ab Seite 32) nachzulesen.

Bevor der Pseudocode der Klasse `DRMapMatching` dargestellt wird, soll noch eine Methode wiedergegeben werden, die in den Methoden der Implementierung an mehreren Stellen benutzt wird. Die Methode `findRelPosition` dient dazu festzustellen, wie ein gegebener Punkt P bezüglich einer Kartenkante L liegt. Liegt P zwischen A und B, dann wird 0 zurückgegeben. Liegt L hinter B, dem Endpunkt der Kante, ist das Ergebnis 1. Ansonsten beträgt der Rückgabewert -1, wenn P vor dem Startpunkt A liegt.

```

findRelPosition (Link L, Position P) {
    if (P between L.A and L.B) return 0;
    if (P after L.B ) return 1;
    if (P before L.A ) return -1;
}

```

Nachfolgend ist der Kopf der Implementierung der Klasse DRMapMatching dargestellt:

```

class DRMapMatching extends DRAlgorithm {
    OID          myOID;
    double       maxDev;
    int          usedPoints;
    double       m_max;

    Position[]  lastPos;
    long[]      lastTimes;

    PosUpdate   lastUpdate = null;
    Link        currLink = null;
    Link        currExpectedLink = null;
    Position    currCorrPos = null;
    Position    lastP = null;
    long        last_time;

    [...]
}

```

Wiedergegeben sind die Attribute der Klasse. Die erste Gruppe besteht neben der ID des Objektes aus den Parametern für das Verfahren: `maxDev` für die maximale Differenz  $d_{max}$  zwischen Soll- und Ist-Position, `usedPoints` für die Anzahl der Punkte zur Berechnung der Regressionsgeraden (zur Bestimmung der Bewegungsrichtung) und `m_max` zur Festlegung des Toleranzbandes um eine Kartenkante.

Die folgende Gruppe dient dazu, die Ist-Positionen in einem Array und die zugehörigen Zeitpunkte zu speichern. Diese beiden Arrays dienen als Datenquelle für die Berechnung der Regressionsgeraden zur Richtungsbestimmung.

Die letzte Gruppe schließlich dient zur Speicherung der Zustandsdaten für den Algorithmus: `lastUpdate` speichert die letzte Updatenachricht, `currLink` referenziert die Kartenkante, die zur letzten Ist-Position gehört, `currExpectedLink` zeigt auf die Kartenkante, die zur Soll-Position gehört, `currCorrPos` deutet auf die mit der Kante `currLink` gematchte Position und `lastP` enthält die letzte Soll-Position.

Zur Festlegung der Parameter des Verfahrens dient die Methode `init`. Es werden lediglich die Parameter in die internen Attribute übernommen.

```

init(theOID, double maximumDev, int usedPts, double mmax) {
    myOID          = theOID;
    maxDev         = maximumDev;
    usedPoints     = usedPts;
    m_max         = mmax;

    max = 0;
    min = 0;
}

```

---

```

}
```

Um die Ist-Position auf eine Kartenkante abzubilden wird die Methode `correctCurrPos` benötigt. Als Parameter wird die Ist-Position erwartet und der Rückgabewert ist ein Punkt auf einer Kartenkante.

```

Position correctCurrPos( Position PM ) {
    if ( currLink == null ) {
        initialize( PM );
    }

    currCorrPos = findNearestLinkPoint( currLink, PM );

    if ( |PM-currCorrPos| > m_max ) then {
        newLink = findNewLink( currLink, PM );
        currCorrPos = findNearestLinkPoint( newLink, PM );
        currLink = newLink;
    }

    // store PM in Position set!
    max = max + 1;
    lastPos[max] = PM;
    lastTimes[max] = current_Time();
    if ( max-min > usedPoints ) {
        min = min+1;
    }

    return currCorrPos;
}

```

Ist das Attribut `currLink` gleich `null`, muß der Zustand initialisiert werden. Das erfolgt durch den Aufruf der Methode `initialize`. Hier wird ausgehend von der Position eine Kartenkante gesucht, die am besten paßt. Die Initialisierung ist bereits im Kapitel 6.2.3 vorgestellt worden, so daß der Code nicht wiederholt wird. Es sei nur angemerkt, daß die Initialisierung auch die Bewegungsrichtung berücksichtigt. Das geschieht dadurch, daß aus den Punkten des Arrays `lastPos` eine Regressionsgerade berechnet wird, welche die Bewegungsrichtung repräsentiert! Nur wenn es nicht möglich ist, eine Richtung zu berechnen (also, wenn weniger als 2 Punkte in `lastPos` sind), wird diese nicht benutzt.

Ist `currLink` initialisiert, wird der zur Ist-Position `PM` der nächste Kantenpunkt `currCorrPos` gesucht. Ist die Entfernung vom `PM` zu `currCorrPos` größer als  $m_{max}$ , wird über die Methode `findNewLink` eine neue Kante gesucht, zu der gematcht wird. Die Methode `findNewLink` wurde übrigens auf Seite 68 im Kapitel 6.2.4.2 dargestellt.

Schließlich wird die Ist-Position im Array abgelegt, das für die Berechnung der Bewegungsrichtung verwendet wird, und die korrigierte Position wird zurückgegeben.

Die folgende Methode, `computeCurrPos`, erfüllt die Aufgabe, Soll-Positionen zu berechnen.

```

Position computeCurrPos() {
    if ( currUpdate != null ) {
        // compute distance from position of last update
    }
}

```

```

deltaT = current_Time() - last_time;
deltaS = currUpdate.v * deltaT;

if ((lastP is within link from A to Bmax) AND
    (distance(lastP,Bmax) >= deltaS) ) {
    lastP = lastP +
        deltaS in direction of currExpectedLink
} else {
    relPos = findRelPosition(currExpectedLink,lastP);
    if ( relPos == 0 ) {
        deltaS = deltaS -
            distance(lastP,currExpectedLink.B);
    } else {
        deltaS = deltaS +
            distance(lastP,currExpectedLink.B);
    }
}

tmpLink = currentExpectedLink;
finished = false;
do {
    tmpLink = chooseSuccLink( tmpLink );

    if ( tmpLink != null ) {
        if ( tmpLink.length()+mmax >= deltaS ) {
            lastP = tmpLink.A +
                deltaS in link's direction
            currentExpectedLink = tmpLink;
            finished = true;
        } else {
            deltaS = deltaS - tmpLink.length();
        }
    } else {
        finish = true;
        lastP = null;
    }
} while (!finished)
}

last_time = current_Time();
return lastP;
} else {
    // here: no update to work with!
    last_time = current_Time();
    return null;
}
}

```

Zunächst wird geprüft, ob bereits ein Update vorgelegen hat. Fehlt es, kann mangels Daten keine Soll-Position berechnet werden. In diesem Fall wird `null` zurückgegeben, um dem Aufrufer zu signalisieren, daß keine Soll-Position errechnet werden konnte.

Ansonsten wird zuerst die Strecke `deltaS` berechnet, welche das Mobile Objekt anhand der Geschwindigkeitsinformation und der verstrichenen Zeit seit dem letzten Update zurückgelegt haben müßte.

Liegt nun der Punkt `lastP` inklusive der Strecke `deltaS` in Bewegungsrichtung noch innerhalb der Strecke vom Kantenanfangspunkt  $A$  zum erweiterten Endpunkt  $B_{max}$  (siehe Abbildung 25 auf Seite 63), ist die neue Soll-Position gefunden. Sonst muß eine Ausgangskante der momentanen Kante gewählt werden, um die Soll-Position zu berechnen.

Vorher muß aber `deltaS` korrigiert werden, je nachdem ob `lastP` hinter dem Endpunkt der Kante  $B$  oder noch vor  $B$  liegt. Ist `lastP` vor  $B$ , muß die Reststrecke bis  $B$  von `deltaS` abgezogen werden, ansonsten muß die Entfernung addiert werden.

Die folgende *do-while*-Schleife dient dazu, eine Nachfolgekante zu bestimmen. Zunächst sucht die Methode `chooseSuccLink` eine Nachfolgekante aus. In unserer Implementierung (die hier nicht wiedergegeben ist), wird stets diejenige Kante ausgesucht, die zur Momentanen den geringsten Richtungsunterschied besitzt. Die Begründung dafür ist die Annahme, daß sich das Mobile Objekt mit höherer Wahrscheinlichkeit weiter Entlang der selben Straße weiterbewegt. Da wir in unserer Datenstruktur keine Unterscheidung zwischen Haupt- und Nebenstraße haben, wird darüber hinaus angenommen, daß die Kante mit dem geringsten Richtungsunterschied die Fortsetzung der benutzten Straße darstellt. Diese Annahme trifft in den meisten Fällen zu, auch wenn abbiegende Vorfahrtsstraßen dadurch nicht erfaßt werden, wenn es eine tangential abgehende Abzweigung gibt. Alternativ könnte man sehr leicht einen anderen Algorithmus zur Auswahl der Folgekante implementieren!

Ist die Nachfolgekante länger als die Strecke `deltaS`, wird diese beibehalten und der Punkt `lastP` ausgehend vom Startpunkt der Kante  $A$  in der Entfernung `deltaS` festgelegt. Die Schleife terminiert. Ist die gefundene Kante zu kurz, wird die Kantenlänge von `deltaS` abgezogen und erneut eine Nachfolgekante gesucht. (Die Schleife geht in die nächste Iteration.)

Schließlich wird der `lastP` zurückgegeben.

Zur Prüfung, ob Soll- und Ist-Positionen bereits um mehr als dem maximalen Abstand voneinander abweichen, gibt es die Methode `deviationExceeded`. Es wird lediglich der Abstand berechnet und entsprechend mit `true` oder `false` geantwortet, wenn der Abstand überschritten oder noch unterschritten ist.

```

boolean deviationExceeded() {
    if (distance(lastPos[max], computeCurrPos()) > maxDev) OR
        (currUpdate == null) {
        return true;
    } else {
        return false;
    }
}

```

Die Methode `computeNewUpdate` erzeugt bei Bedarf Updatenachrichten für den Server. Wird die Methode zum ersten Mal aufgerufen, ist `lastUpdate` noch uninitialized. Das bedeutet, daß auch noch nicht genügend Daten zur Verfügung stehen, um eine Updatenachricht mit kompletten Inhalt zu verschicken, weil erst eine einzige Ist-Position

gespeichert ist. Daher kann keine Geschwindigkeit des Mobilien Objektes berechnet werden! Die Updatenachricht enthält daher 0 als Betrag der Geschwindigkeit.

Im Normalfall wird dann die Geschwindigkeit aus der zurückgelegten Strecke der gespeicherten Punkte und der verstrichenen Zeit bestimmt.

```

PosUpdate computeNewUpdate() {
    if (lastUpdate==null) {
        lastUpdate = new PosUpdate(myOID,
                                   last_time,
                                   currCorrPos,
                                   currLink.getOrientation(),
                                   0 ); // speed

        currExpectedLink = currLink;
        lastExpectedPos = currCorrPos;
    } else {
        if((lastP==null) OR (deviationExceeded())) {
            v = Distance(lastPos[max],lastPos[min])/
                (lastTimes[max]-lastTimes[min]);

            lastUpdate = new PosUpdate(myOID,
                                       last_time,
                                       currCorrPos,
                                       currLink.getOrientation(),
                                       v);

            lastExpectedPos = currCorrPos;
            currExpectedLink = currLink;

            return lastUpdate;
        } else {
            return null;
        }
    }
}

```

Anzumerken ist, daß obwohl der Map-Matching-Algorithmus angewendet wird, nicht die Kante in den Updates übertragen wird. Es genügt vollkommen, die auf die Kante abgebildete Position zu übertragen, weil diese bereits auf einer Kante liegt. Gegebenenfalls wäre es natürlich möglich, auch die Kante explizit im Update zu benennen, wenn dies gewünscht sein sollte.

Der Vollständigkeit halber wird noch die Methode `setNewUpdate` wiedergegeben, die vom Server benutzt wird, um der Algorithmus-Instanz eine vom Client erhaltene Updatenachricht mitzuteilen.

```

setNewUpdate( PosUpdate newUp ) {
    lastUpdate = newUp;
}

```

# 7 Simulation

---

Die beiden in den Kapiteln 5 und 6 vertieften Verfahren, lineares Fortsetzen und kartenbasierte Koppelnavigation, wurden für die Durchführung einer Simulation implementiert. Im nachfolgenden Kapitel wird die Simulationsumgebung und die Simulationsdurchführung näher beschrieben. Anschließend werden die Ergebnisse präsentiert. Schließlich werden die Ergebnisse insbesondere auch in Hinblick auf weitere Verbesserungen der Resultate diskutiert.

## 7.1 Datenerfassung

Ausgangspunkt für die Simulation sind sogenannte Trace-Files. Dabei handelt es sich um Protokolle der Aufenthaltsorte eines Mobilobjektes. Dazu stand ein DGPS-Sensor zur Verfügung, der an ein Notebook angeschlossen wurde. Die vom Sensor gelieferten Positionsdaten wurden für eine spätere Simulation einmal pro Sekunde protokolliert. Anschließend wurden diese Traces in das Simulationsprogramm geladen und mit verschiedenen Parametern mit den beiden Verfahren simuliert.

Die Traces wurden dabei getrennt nach den Klassen Stadt-, Überland-, Autobahnfahrt und Fußgänger erfaßt. Stadtfahrten sind Traces, die mit einem PKW im Stadtverkehr, also innerhalb geschlossener Ortschaften, aufgenommen wurden. Die Fahrten außerhalb der Ortschaften sind im Bereich Überlandfahrten erfaßt. Dabei wurden Bundes-, Landes-, Kreisstraßen, etc. befahren. Da Überlandstraßen immer wieder kleinere Gemeinden und Dörfer durchqueren, ist ein kleiner Teil der Überlandtraces entsprechend auch bei der Durchfahrt dieser Ansiedlungen entstanden. Das entspricht aber dem üblichen Verhalten des Verkehrs, weil es nicht immer Ortsumfahrungen gibt. Deshalb gab es keinen Grund, diese kurzen Ortsdurchfahrten aus den Traces herauszuschneiden. Bei den Autobahnen wurde das Auf- und das Abfahren auf bzw. von der Autobahn nicht in den Traces erfaßt. Nur die Fahrt auf der Autobahn ist enthalten. All diese Traces wurden mit einem PKW aufgenommen.

Zusätzlich wurden noch sogenannte Fußgängertraces aufgenommen, bei denen eine Person zu Fuß mit dem GPS-Gerät und einem Notebook unterwegs war. Auch diese Traces wurden simuliert.

Insgesamt wurden Strecken in einer jeweiligen Gesamtlänge von etwa 89km in Ortschaften, 99km Überland und 163km auf Autobahnen zur Simulation herangezogen. Dazu kamen noch etwa 10km zu Fuß. Insgesamt summiert sich die Gesamtzeit der Traces (ohne Vor- und Nachbearbeitungszeit) auf fast 8 Stunden. Eine Übersicht ist der folgenden Tabelle zu entnehmen.

	<i>Länge</i>	<i>Dauer</i>	<i>Durchschnitts- geschwindigkeit</i>
Autobahn	163km	1h 35min	103km/h
Überland	99km	1h 39min	60km/h
Stadt	89km	2h 25min	34km/h
Fußgänger	10km	2h 08min	4,6km/h

Die Ergebnisse der Simulation für alle Klassen von Traces sind im Kapitel 7.4 ab Seite 87 geschildert.

## 7.2 Kartendaten

Für das Kartenbasierte Verfahren standen Kartendaten der Firma NavTech [NAV] für den Bereich Baden-Württemberg zur Verfügung. Das Datenformat der Daten ist ein proprietäres Format von NavTech mit der Bezeichnung SIF+, zu welchem uns die Spezifikation vorlag.

Da die Daten – wie beispielsweise GDF – nicht nur die hier benötigten Kanten (Links) und Knoten (Nodes) enthalten, sondern darüber hinaus noch diverse Zusatzinformationen, mußten die benötigten Strukturen aus der Masse an Daten herausgelesen werden. Hierzu wurden die SIF+-Daten durch einen selbstgeschriebenen Parser eingelesen und dann die Kanten und Knoten in einem eigenen Format abgespeichert, um die Daten für die Simulation in einer akzeptablen Zeit einlesen zu können.

## 7.3 Simulationsumgebung

Zur Durchführung der Simulation wurde eine eigene Testumgebung implementiert. Als Sprache wurde Java eingesetzt.

Als Referenz zum Vergleich der erzielten Ergebnisse wurde neben den beiden Verfahren Lineares Fortsetzen (Kapitel 5, Seite 45) und Kartenbasierte Koppelnavigation (Kapitel 6, Seite 55) zusätzlich ein Verfahren implementiert, das keine Koppelnavigation verwendet. Dabei handelt es sich um ein sogenanntes „Distance-Based Reporting Protocol“, bei dem das Mobile Objekt immer dann ein Update sendet, wenn es sich um die Entfernung  $d_{max}$  vom letzten Punkt weiterbewegt hat. (Näheres dazu in [LEO01].) Informationen zu Geschwindigkeiten und Bewegungsrichtung benötigt dieses Verfahren nicht.

Das implementierte Simulationsprogramm ermöglicht es, ein Trace-File mit den GPS-Daten zu laden, und eine Simulation durchzuführen. Für das Kartenbasierte Verfahren ist es zudem möglich, eine Datei mit einer Karte zu laden. Für die Simulation stehen drei Verfahren zur Auswahl: Distance-Based Reporting Protocol, Lineares Fortsetzen und Kartenbasierte Koppelnavigation. Für jedes Verfahren lassen sich die Parameter über Menüs festlegen (z.B.  $d_{max}$ , Anzahl der Punkte zur Bestimmung einer Bewegungsrichtung, etc.). Das Simulationsergebnis wird dann sowohl in Form einer Statistik ausgegeben als auch graphisch dargestellt. Dadurch ist zum einen möglich, die Zahlenwerte für die statistische Auswertung

aufzunehmen. Zum anderen kann man sich sehr einfach ein Bild davon machen, welchen Weg das Mobile Objekt zurückgelegt hat und wann das jeweilige Verfahren jeweils ein Update versendet. Der Inhalt der Nachricht wird für den Benutzer graphisch dargestellt.

Nachfolgend werden ein paar herausragende Kernpunkte des Simulators näher erläutert. Das sind sowohl Themen, die das GUI betreffen als auch Implementierungsdetails.

### 7.3.1 Strukturierung des Simulators

Kerngedanke bei der Implementierung der Simulationsumgebung war die Aufteilung nach dem Model-View-Controller-Konzept. Insbesondere ist zwischen dem GUI, das für die Eingabe der Parameter und die Präsentation der Ergebnisse verantwortlich ist, und dem eigentlichen Simulator zu unterscheiden, der die Logik implementiert, um von der Eingabe zum Ergebnis zu gelangen.

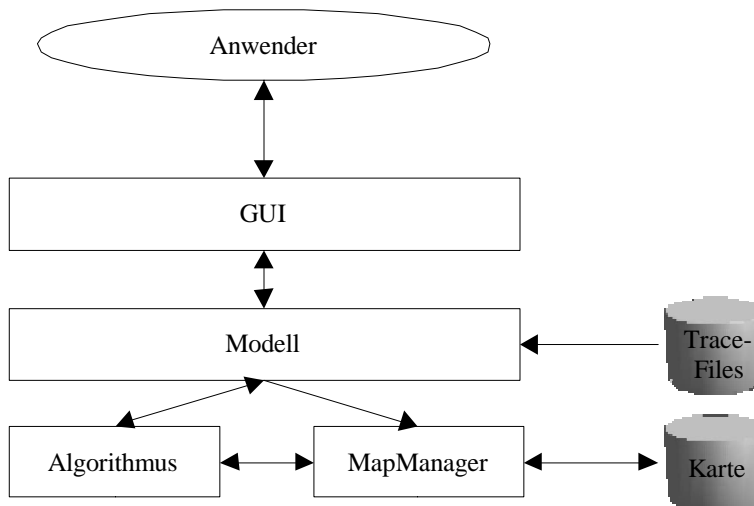


Abbildung 28 – Überblick über die Hauptkomponenten des Simulators

Wie in Abbildung 28 zu sehen, besteht die Logik des Simulators aus den Hauptkomponenten Modell, Algorithmus und MapManager. Das Modell ist die Komponente, die mit dem GUI interagiert. Es nimmt die Eingaben des Benutzers entgegen und ist verantwortlich für die Durchführung der Simulation. Dazu lädt es ein Tracefile, das vom Benutzer ausgewählt wurde, erzeugt eine Instanz einer Algorithmusklasse (siehe Kapitel 4.2) und gibt dem MapManager den Auftrag, eine bestimmte Karte zu laden, wenn dies erforderlich ist.

Während die Simulation läuft übergibt das Modell nach und nach jede einzelne Position aus dem Tracefile an den Algorithmus. Die Positionen dienen dem Algorithmus als Eingabe der Ist-Positionen. Darauf hin berechnet der Algorithmus die Soll-Position und erzeugt bei Bedarf Updatenachrichten, die an das Modell übergeben werden. Die Vorgehensweise entspricht dabei dem Ablauf, der im Kapitel 4.4 dargestellt ist. Das Modell bereitet dann auch die Daten auf, um sie vom GUI anzeigen zu lassen.

Der Map Manager erfüllt zwei Aufgaben. Zum einen ist er dafür zuständig eine Karten-Datei zu laden, so daß der kartenbasierte Algorithmus Zugriff auf die Daten hat. Zum anderen ist

der Map Manager dafür zuständig, aus den SIF+-Daten die Daten für die Knoten und Kanten für einen bestimmten geographischen Bereich herauszulesen und in einer separaten Datei zu speichern. Nur so ist es möglich, eine Karte in annehmbarer Zeit für eine Simulation zu laden, weil die ganze SIF+-Datei<sup>10</sup> viel zu groß wäre.

Beim kartenbasierten Verfahren interagiert die Algorithmus-Instanz mit dem Map Manager, um die benötigten Knoten und Kanten der Karte zu bekommen.

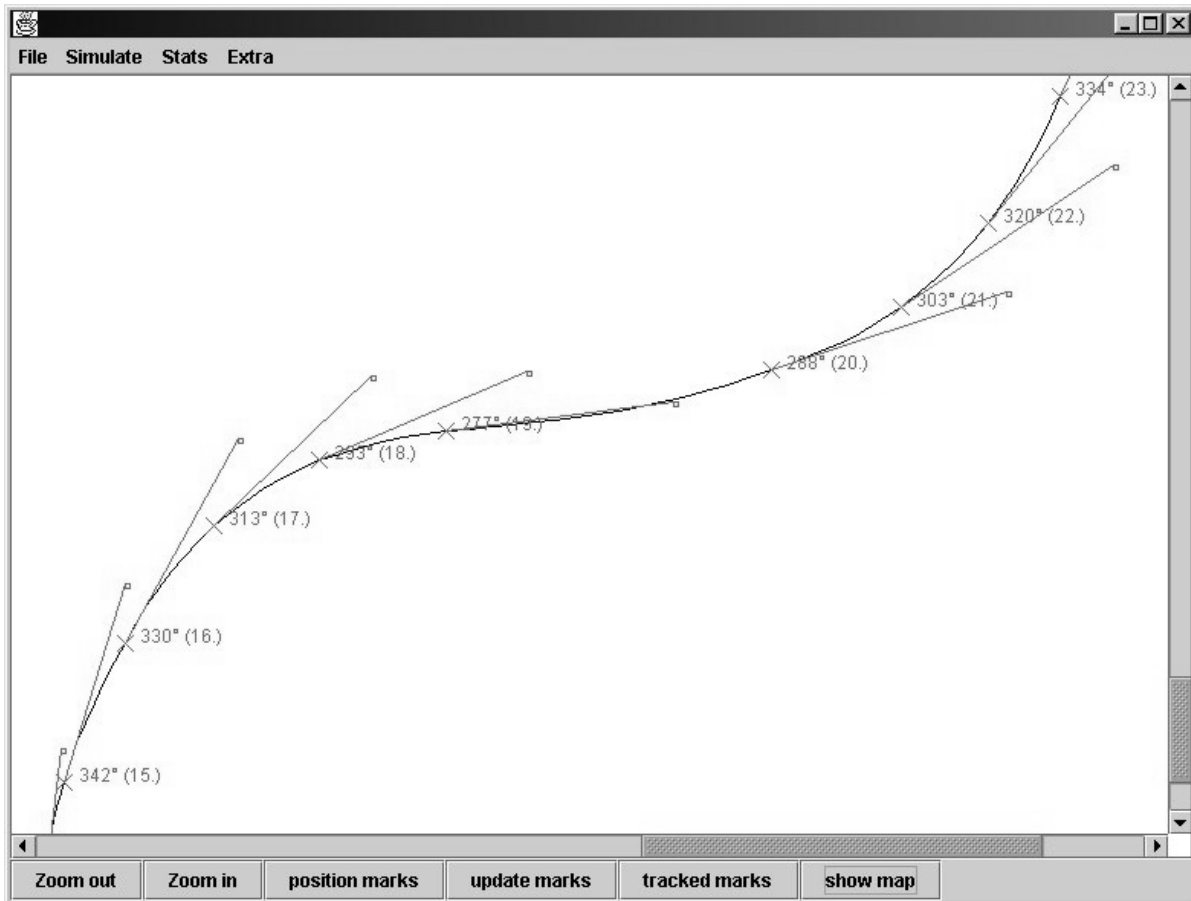


Abbildung 29 – Screenshot des GUIs des Simulators von einer Autobahnfahrt. Zu sehen sind die Abgefahrenen Positionen des Fahrzeugs (schwarze Linie) und die verschickten Lageupdates (rot). Verwendet wurde das Verfahren Lineares Fortsetzen.

Genau genommen arbeitet der Simulator wie das Programm auf einem Mobilien Objekt. Ausgehend von den Ist-Positionen (des Trace-Files) werden die Soll-Positionen und die Updatenachrichten errechnet. Die Serverseite und damit auch die Kommunikation zwischen dem Client und dem Server sind nicht Teil der Simulation.

Um aber die Performance der einzelnen Algorithmen festzustellen reicht dieser Aufbau bereits aus, weil das Ziel, die Kommunikation und damit die Anzahl an Updatenachrichten zu minimieren, anhand des obigen Aufbaus gemessen werden kann.

<sup>10</sup> Die uns zur Verfügung gestellte Datei enthält die Daten für Baden-Württemberg und ist ca. 1GByte groß.

### 7.3.2 Struktur der Kartendaten

Wie bereits angedeutet wurden die Kartendaten nicht direkt aus der SIF+-Datei geladen, sondern in ein eigenes Format überführt. Bei der Konvertierung ist es möglich, einen Bereich festzulegen, für den die Daten aus der SIF+-Datei extrahiert werden sollen. Nur diese werden dann in der neuen Datei abgespeichert. Der Grund dafür ist, daß die gesamte SIF+-Datei zu groß gewesen wäre, um sie im Hauptspeicher zu behalten und das Laden sehr lange gedauert hätte.

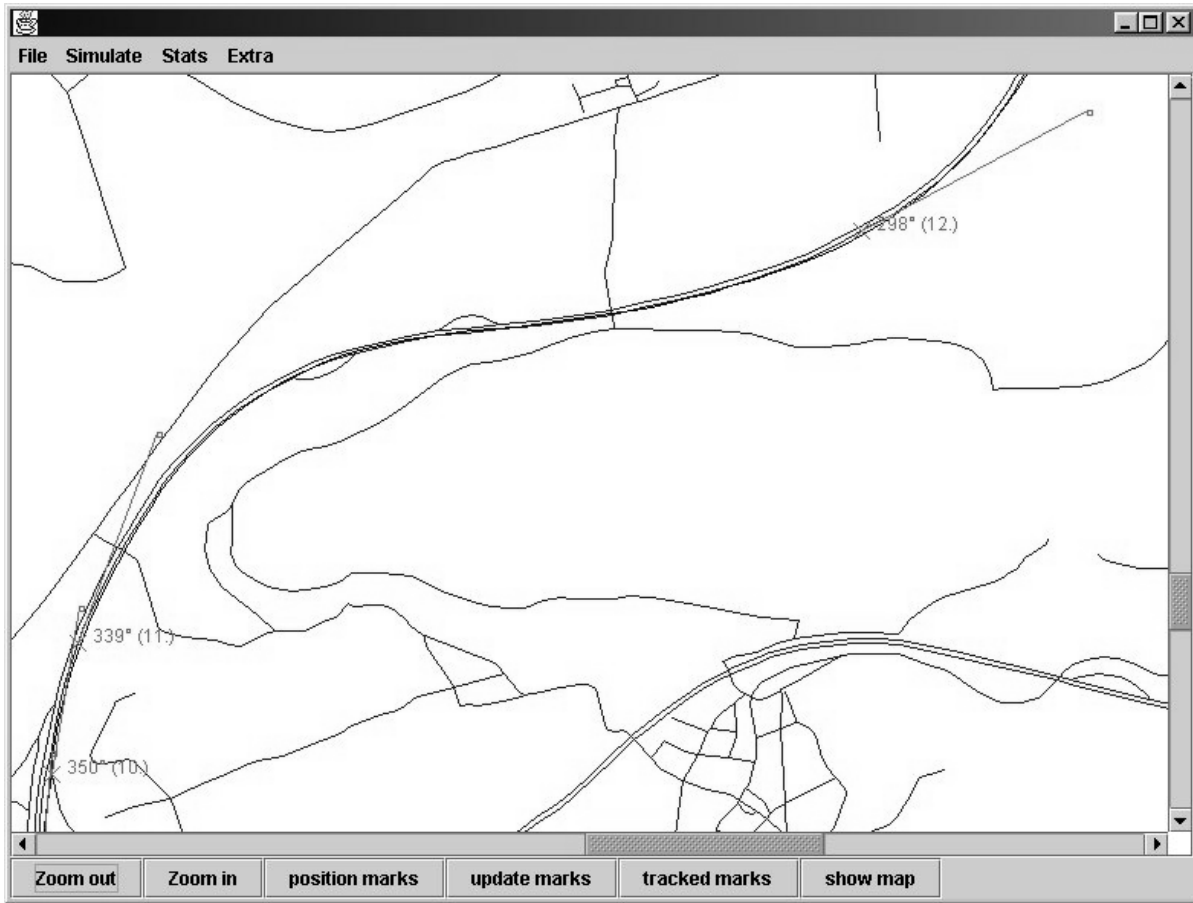


Abbildung 30 – Screenshot des Simulators bei der Simulation der gleichen Autobahnfahrt wie in Abbildung 29 allerdings unter Einsatz des Map Matching Verfahrens. Alle Parameter sind sonst gleich. Man kann erkennen, daß deutlich weniger Updates benötigt werden.

Durch die Beschränkung auf einen bestimmten Bereich, wurde die Datenmenge auf die benötigte Umgebung reduziert. Außerdem wurden sämtliche Daten aus der SIF+-Datei bei der Konvertierung weggelassen, die nicht für die Simulation benötigt wurden. Dazu gehören beispielsweise Straßennamen, „Points of Interest“, etc. Es werden lediglich Kanten (Links) und Knoten (Nodes) aus der SIF+-Datei übernommen.

Um einen effizienten Zugriff auf die geladenen Knoten und Kanten der Karte zu erlangen, werden beim Laden einer Karte drei Indexstrukturen im Speicher aufgebaut: ein Index nach Knoten-IDs, ein Index nach Kanten-IDs und ein räumlicher Index für die Knoten.

Die beiden Indizes, die eine Suche nach IDs erlauben, sind als Binärbäume realisiert. Dabei kam die Klasse `java.util.TreeMap` zu Einsatz, die einen Red-Black-Baum verwaltet und zum standard Java-JDK gehört. Zugriffe wie Einfügen, Löschen oder Suchen werden in  $O(\log(n))$  Schritten durchgeführt. Dabei ist  $n$  die Anzahl der Elemente im Baum.

Der räumliche Index auf den Knoten ist als Quadtree realisiert. Die Implementierung war bereits im NEXUS-Programmpaket enthalten. Mit dem Index ist es möglich, auf effiziente Weise alle Knoten in einem Umkreis um eine bestimmte Koordinate zu bestimmen. Diese Art von Suche wird bei der Initialisierung benötigt (siehe Kapitel 6.2.3).

Sowohl das Laden von Karten, die Verwaltung und der Aufbau der Indexstrukturen als auch die Konvertierung von SIF+ ins eigene Format werden vom MapManager erledigt. Andere Komponenten des Simulators haben Zugriff auf die Knoten und Kanten über die Methoden der Klasse, so daß komplettes Information Hiding umgesetzt wird, um die Konsistenz der Daten stets zu erhalten.

### 7.3.3 Graphische Datenpräsentation

Zur anschaulichen Visualisierung der Ergebnisse eines Algorithmus und zur Beurteilung einzelner Situationen wurde in den Simulator eine Komponente integriert, die die Resultate der Simulation graphisch darstellt.

Beispiele dafür sind in den Abbildungen 29, 30 und 31 zu sehen. Bei Abbildung 29 ist das Lineare Fortsetzen abgebildet, während Abbildung 30 eine kartenbasierte Simulation wiedergibt. Man kann hier erkennen, daß die Darstellung der Karte eingeschaltet ist. Abbildung 31 zeigt das Distance-Based-Reporting-Protokoll. Alle Darstellungen zeigen den selben Streckenbereich unter Verwendung des selben Trace-Files und der selben Parameter für die Simulation. Man kann bereits erkennen, daß das kartenbasierte Verfahren viel weniger Upates verschicken muß.

Die schwarze Linie zeigt in den Abbildungen die aufgezeichneten Positionen des Trace-Files. Die diskreten Punkte werden dabei als Polygonzug dargestellt. Wenn gewünscht, lassen sich aber auch die einzelnen Positionspunkte („Position Marks“) anzeigen. Dann erscheinen sie als schwarze Kreuze auf der Linie.

Updatenachrichten werden rot dargestellt. Die Positionsmeldung, die in einem Update verschickt wird, ist durch ein Kreuz an der betreffenden Stelle dargestellt. Die Richtung und Geschwindigkeit, die dem Server mitgeteilt werden, werden durch die Richtung und die Länge der vom Kreuz ausgehenden Linie modelliert. (Das kleine Rechteck am Ende der Linie dient lediglich zur leichteren Wiedererkennung des Endes der Linie.) Auch die Updatenachrichten („Update Marks“) lassen sich wie die Positionspunkte über einen Button ein- und ausschalten.

Nicht in den Abbildungen zu sehen, aber auch zuschaltbar, sind die Soll-Positionen („Tracked Marks“), die der jeweiliger Algorithmus berechnet hat. Auch diese werden durch Kreuze dargestellt.

Ist eine Karte geladen, läßt sie sich ebenfalls ein- und ausblenden. Das geschieht über den Knopf „Show Map“. In der Abbildung 30 ist die Karte eingeschaltet.

Als zusätzliche Funktion ist das Zoomen implementiert worden, um dem Benutzer zu

ermöglichen, sich sowohl einen Überblick über die Simulation zu verschaffen als auch einzelne Teile im Detail zu betrachten.

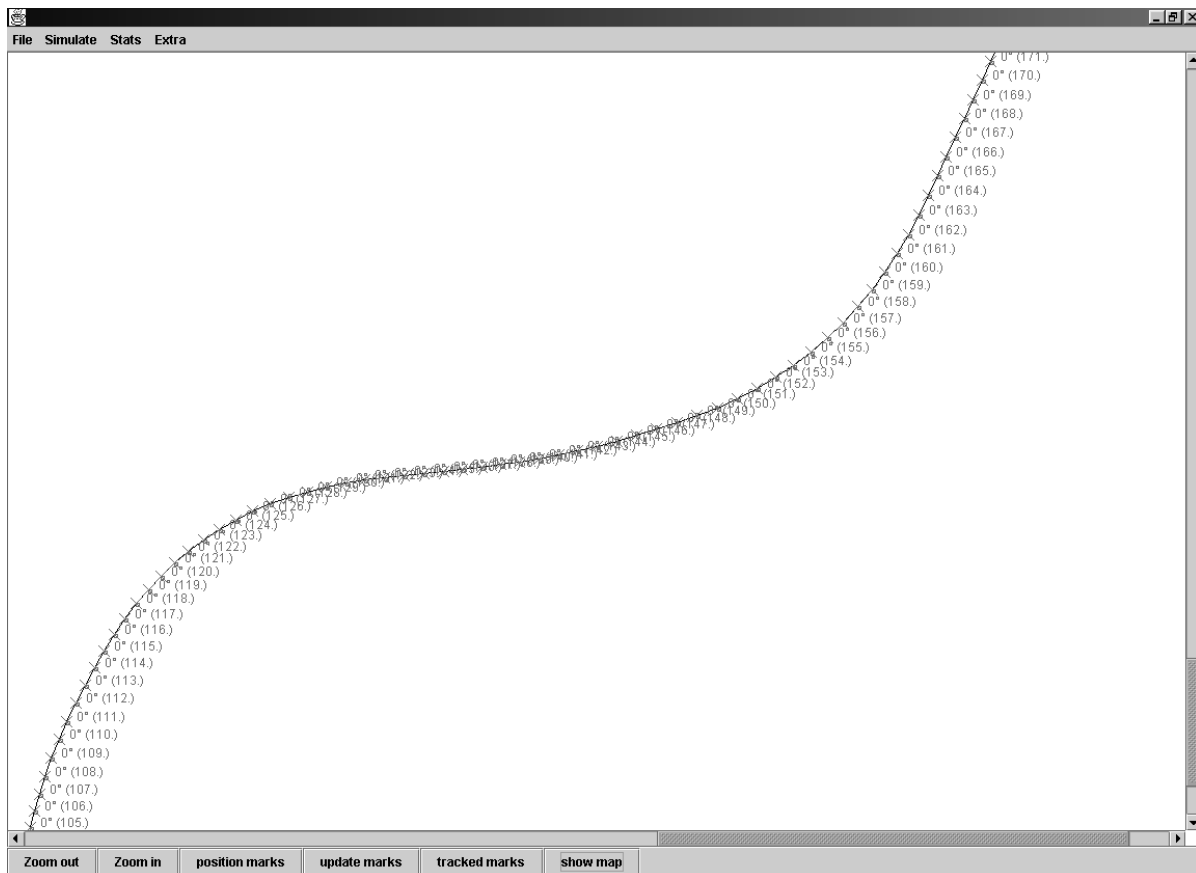


Abbildung 31 – Erneut der gleiche Streckenausschnitt wie in den beiden vorangegangenen Abbildungen. Dieses Mal kam das Nicht-Dead-Reckoning-Verfahren bei gleichem Parameter  $d_{max}$  zum Einsatz. Es ist deutlich zu erkennen, daß viel mehr Updates verschickt werden.

Die Aufbereitung der graphischen Darstellung erfolgt im Wesentlichen durch zentrische Streckung der Koordinaten der Punkte, so daß sie in den gewünschten Bereich des Bildschirms passen. Zusätzlich ist eine Translation erforderlich, um dem Punkt mit den geringsten x- und y-Koordinaten auf den Ursprung des Fensters abzubilden.

### 7.3.4 Kenndaten der Simulation

Bevor im nächsten Kapitel die Ergebnisse der Simulation präsentiert werden, sollen ein paar Kenndaten zur Hardware und zur Simulation vermittelt werden.

Die Simulation erfolgte auf einem Rechner mit einem AMD Athlon Prozessor mit 700MHz. Als Hauptspeicher standen 256MB zur Verfügung. Das in Java geschriebene Programm lief dabei auf einer Virtual Machine von Sun in der Version 1.3.

Eine Simulation eines Trace-Files mit dem Map Matching Algorithmus dauert etwa 2-3 Sekunden. Beim Linearen Fortsetzen erfolgt eine Simulation in der Größenordnung von einer Sekunde.

Das Laden einer Karte des Großraumes Heilbronn, die eine Fläche von etwa 30x50km abdeckt, dauert etwa 23 Sekunden. Diese Karte besitzt ca. 100000 Knoten und 210000 Kanten. Dabei sind auch die Daten für Städte und Gemeinden enthalten.

## 7.4 Ergebnisse

Die Simulation bestätigte das erwartete Ergebnis, daß das Lineare Fortsetzen gegenüber dem Nicht-Dead-Reckoning-Verfahren weit weniger Updatenachrichten verschicken muß. Erwartungsgemäß spart das Map Matching Verfahren noch einmal Updates gegenüber dem Linearen Fortsetzen ein.

Abbildung 32 zeigt bereits eine grobe Übersicht über die erzielten Resultate für einen Toleranzradius von  $d_{max}=20m$ . Sie soll einen ersten Eindruck davon vermitteln, welche Ergebnisse erzielbar sind.

	<i>Lineares Fortsetzen</i>		<i>Map Matching</i>		<i>Distance Based Reporting</i>	
<i>Autobahnfahrt</i>	22,30%	520 Updates/h	15,80%	367 Updates/h	100,00%	2327 Updates/h
<i>Überlandfahrten</i>	21,80%	541 Updates/h	18,80%	454 Updates/h	100,00%	1866 Updates/h
<i>Innerorts</i>	35,80%	422 Updates/h	32,50%	404 Updates/h	100,00%	1138 Updates/h
<i>zu Fuß</i>	33,10%	69 Updates/h	34,30%	67 Updates/h	100,00%	203 Updates/h

Abbildung 32 – Übersicht über die erzielten Ergebnisse der Verfahren bei  $d_{max}=20m$ . Die Prozentzahl entspricht der relativen Anzahl an Updates, die im Vergleich zum Distance-Based-Reporting-Protokoll verschickt wurden.

Achtung: Die Einsparung entspricht 1 minus dem Wert aus der Tabelle!

Man kann bereits feststellen, daß pro Zeiteinheit bei Verwendung von Dead-Reckoning sehr viel weniger Updatenachrichten verschickt werden. Zudem ist zu erkennen, daß je langsamer die Bewegungscharakteristik ist, um so weniger Nachrichten pro Stunde das Distance-Based-Protokoll verschickt. Das liegt daran, daß wegen der geringeren Geschwindigkeit weniger häufig ein Update verschickt werden muß. Wegen der dadurch kleineren absoluten Nachrichtenzahl, ist es für die Dead-Reckoning schwieriger, ihren Vorteil zu halten. Die relativen Zahlen zeigen, daß von der Autobahnfahrt zum Fußgänger ein Anstieg der Prozentzahl bei gleicher Genauigkeit zu verzeichnen ist. Trotzdem ist die Einsparung mit Dead-Reckoning noch sehr hoch!

In den folgenden Teilkapiteln werden die Ergebnisse detaillierter präsentiert. Zunächst geht es um das Lineare Fortsetzen und anschließend um die Kartenbasierte Koppelnavigation.

### 7.4.1 Lineares Fortsetzen

Es ist erstaunlich, daß bereits der einfache Algorithmus des Linearen Fortsetzens gegenüber dem nicht-dead-reckoning Verfahren nur noch 16% bis 44% außerhalb von Ortschaften bzw. bis zu 82% Innerorts an Updates Verschicken muß. Das entspricht einer Einsparung von bis zu 84% an Kommunikation!

Die Schwankungen der Ergebnisse lassen sich auf zwei Gründe zurückführen: Zum einen auf den Toleranzradius  $d_{max}$  und zum anderen auf die Bewegungscharakteristik.

Wie man in den Diagrammen im Anhang (ab Seite 106) erkennen kann, nimmt der Vorteil der Koppelnavigationsverfahren ab, je größer die Strecke  $d_{max}$  gewählt wird. Das liegt daran, daß es mit zunehmender Toleranzschwelle immer schwieriger ist, das Verhalten des Mobilien Objektes für längere Zeit korrekt vorherzusagen. (Je größer  $d_{max}$  wird, desto längere Zeit verstreicht bis diese Strecke zurückgelegt wird. Damit steigt der Zeitraum, für den die Vorhersage des Koppelnavigationsprotokolls zutreffen muß.)

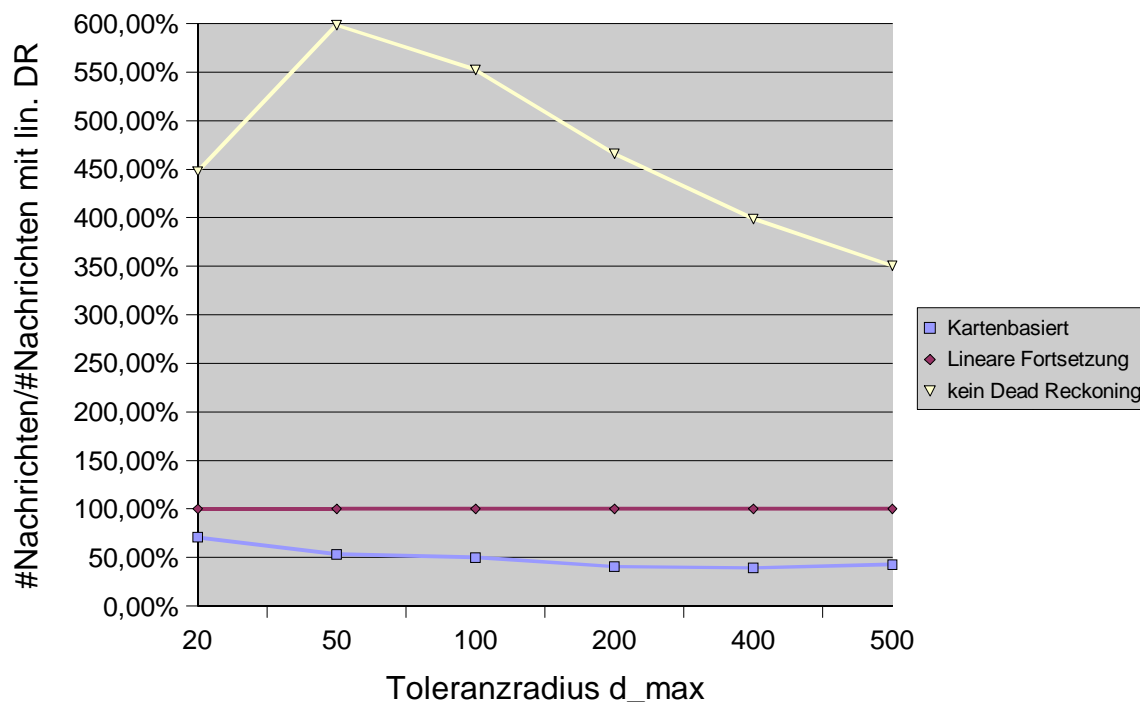


Abbildung 33 – Simulationsergebnisse bei der Autobahnfahrt

Die Bewegungscharakteristik entscheidet in so fern über die Resultate, als daß eine Bewegung auf der Autobahn mit weniger Geschwindigkeits- und Richtungsschwankungen vonstatten geht als bei einer Fahrt in der Stadt. Überlandfahrten liegen zwischen diesen beiden Extrema. Durch gelegentliche Ortsdurchfahrten und das Treffen auf Wegkreuzungen, kommt es zu einer größeren Schwankung der Geschwindigkeit als bei der Autobahnfahrt. Die Straßenführung ist kurvenreicher als bei einer Autobahn. Trotzdem ist die Geschwindigkeit gleichmäßiger als bei der Stadtfahrt. Auch ist die Anzahl der gefahrenen Kurven nicht so hoch wie Innerorts.

Ist die Bewegungscharakteristik recht konstant (wie auf der Autobahn), können die Vorhersagen der Koppelnavigationsprotokolle besser greifen, weil die zukünftige Bewegung besser vorhersagbar ist. Ist die Charakteristik sprunghafter wie auf Überlandfahrten oder besonders in Ortschaften, kann die Vorhersage nicht so häufig greifen. Die Verbesserungen gegenüber einem Nicht-Dead-Reckoning-Verfahren sind folglich geringer.

Zu Fuß spart das Dead-Reckoning-Verfahren bis zu 67% an Nachrichten ein. Auch hier ist wie bei allen anderen Kategorien die Tendenz zu beobachten, daß der Vorteil des Verfahrens sinkt, je größer  $d_{max}$  gewählt wird.

Bei der Simulation ist aufgefallen, daß es für die Autobahn-, Überland und Stadtfahrten am besten ist, wenn 4 Positionspunkte zur Berechnung einer Regressionsgeraden – und damit zur Bestimmung der Orientierung – benutzt werden. Bei den Fußtraces erwiesen sich dagegen 8 Punkte zur Ermittlung der Richtung als Optimum.

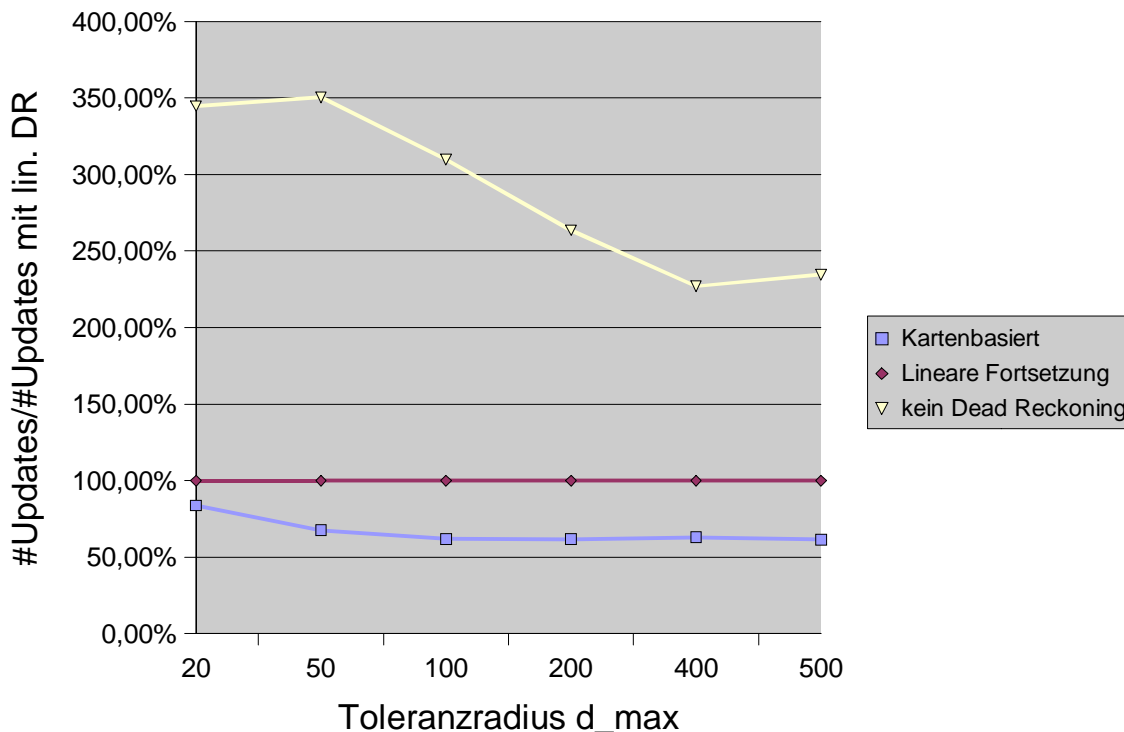


Abbildung 34 – Simulationsergebnisse bei Überlandfahrten

Erklärt wird dies damit, daß auf den Fahrten mit dem PKW bereits wenige Positionspunkte wegen der höheren Geschwindigkeit weit genug auseinander sind, so daß die Ungenauigkeit des DGPS-Gerätes eine geringere Rolle spielt. Mehr Punkte machen den Richtungsvektor träge. Er reagiert zu langsam, wenn sich die Richtung kontinuierlich ändert. Daher ist die Vorhersage der zukünftigen Bewegung des Fahrzeugs häufiger falsch. Es entstehen mehr Updatenachrichten. Die Wahl von weniger als 4 Punkte ist für den Stadtverkehr nicht geeignet. Es entstehen deutlich mehr Updates. Zu Fuß sind die Geschwindigkeiten geringer. Die Ungenauigkeit des Lagemeßgerätes wirkt sich stärker auf den Richtungsvektor aus. Hier ist es nötig, mehr Punkte zu benutzen, um Ungenauigkeiten auszumitteln. Nimmt man zu

viele Punkte, gilt das gleiche wie oben: Der Richtungsvektor reagiert zu träge und mehr Updates kommen zustande. Zu wenige Punkte wirken sich auch negativ auf das Ergebnis aus.

Ein weiterer Hinweis, daß die Geschwindigkeit des Fahrzeugs von Bedeutung ist, findet sich beim Vergleich der Ergebnisse der schnelleren Traces, Autobahn und Überland. Man stellt fest, daß es bei diesen beiden fast keinen Unterschied macht, ob man 4 Positionspunkte oder weniger benutzt. Bei der Wahl von nur 2 Punkten könnte man dann sogar auf die aufwendige Berechnung der Regressionsgeraden verzichten. Trotzdem sind die Resultate nur geringfügig schlechter als mit 4 Punkten! Im Stadtverkehr gilt diese Aussage aber nicht! Hier ist die Geschwindigkeit zu gering, um weniger Stützpunkte für die Richtungsinterpolation zu benutzen.

## 7.4.2 Kartenbasierte Koppelnavigation

Mit Hilfe des Map Matchings wird gegenüber dem Linearen Fortsetzens bei den PKW-Traces eine weitere Verbesserung zwischen 37% und 5% erzielt. Auch hier zeigen sich die gleichen Tendenzen: Auf Autobahnen ist die Verbesserung größer als Innerorts. Zwischen diesen beiden Extrema liegen die Überlandfahrten.

Die Gründe für die Trends sind die gleichen wie beim Linearen Fortsetzens. Auch hier sind die Bewegungscharakteristik und der Toleranzradius entscheidend.

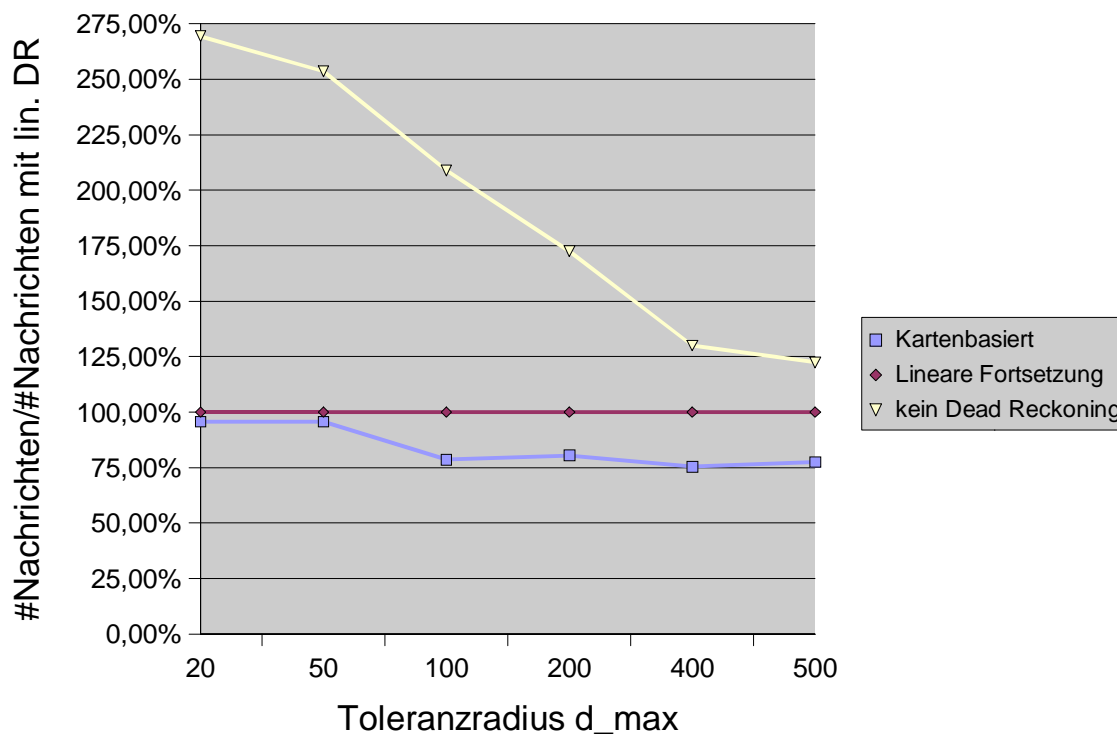


Abbildung 35 – Gegenüberstellung der Verfahren bei Stadtfahrten

Die zusätzliche Verbesserung gegenüber dem Linearen Fortsetzens ist auf den Abgleich mit der Karte zurückzuführen. Dadurch ist es dem Verfahren möglich, Kurvenfahrten vorherzusagen und damit Updatenachrichten einzusparen. Darüber hinaus ist aber

anzumerken, daß es Innerorts schwieriger als außerhalb von Städten ist, die weitere Bewegung korrekt vorherzusagen, weil die Dichte von Abzweigungen größer ist. Damit steigt auch die Wahrscheinlichkeit, daß das Verfahren eine andere Kante wählt als das Fahrzeug. Erschwerend kommt dabei hinzu, daß Innerorts durch die stark schwankende Bewegungscharakteristik häufiger Updates benötigt werden, weil an Ampeln angehalten werden muß, Abzweigungen genommen werden, etc.

Bei den Fußgängern zeigt sich bei  $d_{max}=20m$ , daß das Kartenbasierte Verfahren sogar etwas über 3% schlechter ist als das Lineare Fortsetzen. (Das erzielte Ergebnis spart aber immer noch 65% der Nachrichten des Nicht-Dead-Reckoning-Verfahrens ein!) Der Grund für die Schwäche ist, daß der Kartenabgleich zum Teil falsche Ergebnisse liefert. Dadurch entstehen unnötige Updates. Bei allen anderen simulierten  $d_{max}$ -Werten erzielt das Map-Matching-Verfahren aber eine Verbesserung um bis zu 50% gegenüber dem Fortsetzungsverfahren.

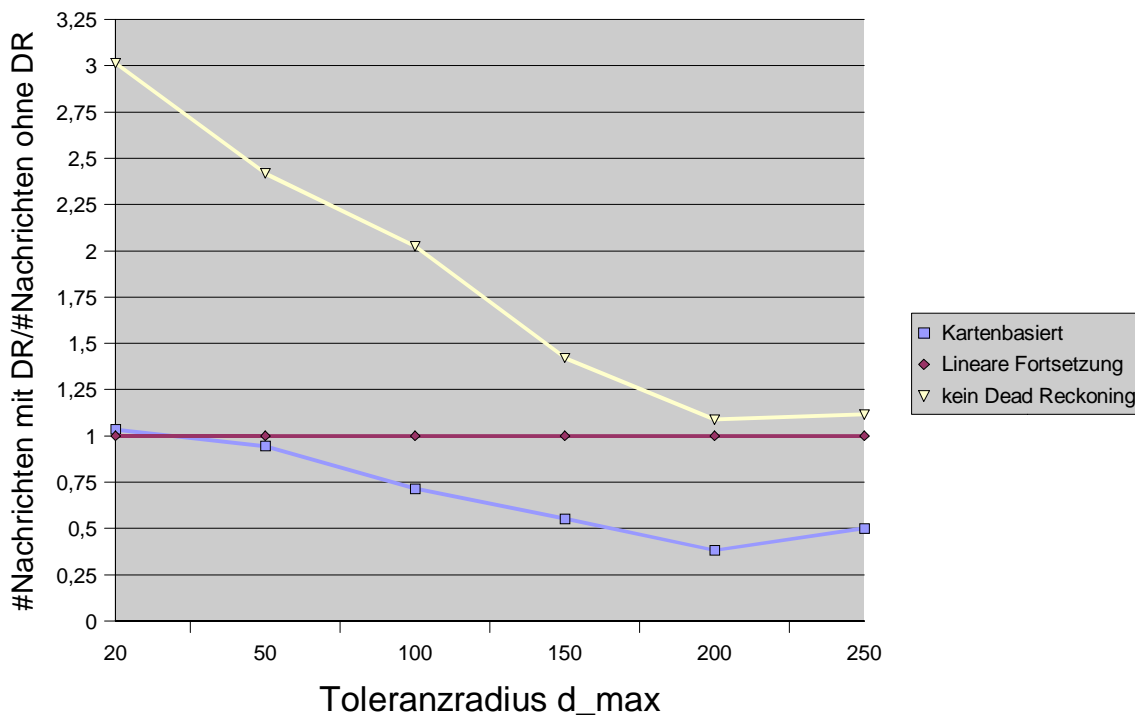


Abbildung 36 – Ergebnisse der Fußtraces. Zu beachten ist, daß hier andere  $d_{max}$ -Werte simuliert wurden. Größere Werte, wie sie bei den anderen Verfahren angewendet wurden, sind hier nicht sinnvoll, weil die einzelnen Traces nicht so lang sind!

Die Anzahl der benutzten Punkte zur Bestimmung der Bewegungsrichtung sind wie im Fall des Fortsetzungsverfahrens gleich. Auf Autobahn-, Überland- und Stadtfahrten sind 4 Punkte am besten, während zu Fuß mit 8 Punkte die besten Resultate zu beobachten waren. Offensichtlich gilt auch hier die gleiche Begründung wie beim anderen Verfahren.

Ein weiterer Parameter des Kartenbasierten Verfahrens ist  $m_{max}$ : Die Entfernung, die abgewartet wird, bevor eine neue Kante gesucht wird. Es war für den Wert nicht möglich, eine feste Kalibrierung festzustellen. Allerdings wurden bei  $d_{max}<100m$  in der Regel die besten Resultate mit  $m_{max}\leq 40m$  erzielt. Höhere  $m_{max}$ -Werte führten zu einer Erhöhung der

Updaterate. Die gewählten Werte sind im Anhang nachzulesen. Zu beachten ist, daß mit steigendem  $d_{max}$  die absoluten Zahlen der Updates eines Verfahrens unabhängig von  $m_{max}$  immer näher beieinander sind. Wenn der Toleranzradius steigt, sinkt der Einfluß von  $m_{max}$  auf das Ergebnis. Das bedeutet, daß die Wahl von  $m_{max}$ -Werte unter  $40m$  nahe am Optimum ist.

Zur Strategie der Wahl der neuen Kante ist zu sagen, daß es sich als falsch erwies, nach einem Knotendurchlauf nur dann neue Kante zu wählen, wenn der Knoten nur eine einzige Ausgangskante besitzt und sonst immer ein Update zu generieren. Die erzielten Ergebnisse waren zum Teil sogar schlechter als die Resultate ohne Map Matching! Vorallem in der Stadt, wo die Knotendichte sehr hoch ist und ein Knoten sehr häufig mehr als nur eine Ausgangskante besitzt, waren die Ergebnisse besonders schlecht.

Die vorliegenden – deutlich besseren – Simulationsergebnisse wurden dagegen mit der Strategie erzielt, stets die Folgekante zu wählen, die zur bisherigen Kante den geringsten Winkelunterschied besitzt. (Eine weiter Verbesserung dürfte wohl auftreten, wenn die Strategie implementiert werden würde, bei der die Kante gewählt wird, welche die momentane Straße fortsetzt.)

### 7.4.3 Bewertung

Anhand der erzielten Ergebnisse ist klar zu erkennen, daß sich der Einsatz von Koppelnavigationsprotokollen bei allen Bewegungsmustern lohnt. Bereits ein einfaches Verfahren wie das Lineare Fortsetzen führt zu einer drastischen Absenkung der Kommunikation. Dabei ist der Implementierungsaufwand für dieses Verfahren nicht sehr hoch! Auch die Anforderungen an die Hardware sind gering. Damit ist das Verfahren überall einsetzbar, wo Ortsinformationen an einen Lokationsserver übertragen werden müssen.

Das Kartenbasierte Verfahren schneidet im Vergleich zum Fortsetzungsverfahren noch besser ab, benötigt dafür aber einen leistungsfähigeren Prozessor beim Client. Das größte Problem dürfte aber die Unterbringung der Karte sein. Auf Grund der Größe der Daten wird das Datenvolumen sehr schnell zu groß für einen PDA mit 8 oder 16MB Speicher<sup>11</sup>. Erst leistungsfähigere Rechner, die mehr Speicher besitzen, können Karten für einen größeren Bereich speichern und verwalten. Das bedeutet aber auch, daß die Geräte größer und schwerer sind. Im Auto, wo es abzusehen ist, daß in Zukunft immer mehr Fahrzeuge Navigationssysteme haben werden, wären die Möglichkeiten gegeben, das Map Matching Verfahren umzusetzen. Das eingebaute Navigationssystem könnte dann sowohl (wie bisher) die Aufgabe der Routenplanung erfüllen als auch als Kartendatenbank für das Koppelnavigationsprotokoll dienen.

Denkbar wäre es auch, neben der oft erwähnten Reduktion des Kommunikationsvolumens, bei gleichem Datenvolumen, ein Mobiles Objekt genauer zu lokalisieren. Statt einer maximalen Ungenauigkeit  $d_{max}=x$  bei einem Nicht-Dead-Reckoning-Protokoll könnte man ein Koppelnavigationsalgorithmus einsetzen, das eine Positionierungsgenauigkeit  $y < x$  besitzt und (auf Grund einer Vorteile) die gleiche Updaterate besitzt. Die Menge der Kommunikation bleibt gleich, aber die Genauigkeit wird durch den Einsatz eines Dead-Reckoning-Protokolls höher.

---

<sup>11</sup> Diese Größenordnung entspricht der zur Zeit üblichen Ausstattung dieser Kleinstcomputer.

## 7.5 Fehlerabschätzung

Nach der Darstellung der Ergebnisse der Verfahren in den jeweiligen Klassen, geht es nun um die Abschätzung, welche Veränderungen bei einer konkreten Realisierung zu erwarten wären. Darüber hinaus soll dargestellt werden, welche Verbesserungen oder Erweiterungen eingebaut werden könnten, um die Resultate weiter zu optimieren.

### 7.5.1 Einordnung anderer Verfahren

In der Simulation haben wir die beiden Verfahren Lineares Fortsetzung und Kartenbasierte Koppelnavigation betrachtet. Nun sollen die zu erwartenden Resultate abgeschätzt werden, welche die anderen Dead-Reckoning-Verfahren liefern würden.

Das Lineare Fortsetzen ist das einfachste Verfahren aller Koppelnavigationsprotokolle. Es nimmt stets eine Fortsetzung der Bewegung an, wie sie bisher stattgefunden hat. Als Bewegungsrichtung wird stets eine Gerade angenommen. An dieser Stelle unterscheidet sich das Verfahren mit Krümmungsapproximation von der Linearen Fortsetzung. Werden Krümmungen bei der Vorhersage zugelassen, können bereits komplexere Bewegungen von Mobilien Objekten modelliert werden. Es ist zu erwarten, daß diese Modellierungsart dem tatsächlichen Verhalten von Objekten näher kommt, weil nicht nur geradlinige Bewegungen, sondern auch Krümmungen (Kurven) beschrieben werden können.

Das Historienbasierte Verfahren geht – wie bereits erwähnt – in ein Kartenbasiertes Verfahren über, weil aus den aufgezeichneten Historien Karten gewonnen werden. Die Performance dürfte zwischen den Fortsetzungsverfahren und dem reinen Kartenbasierten Verfahren liegen, je nachdem, ob sich die Mobilien Objekte eher entlang der Historienpfade oder abseits davon bewegen.

Zum Schluß bleibt noch übrig, die Kartenbasierten Verfahren untereinander einzuordnen. Das Verfahren, welches den Ausgangskanten eines Knotens Wahrscheinlichkeiten zuweist, dürfte wie das Verfahren mit Routenplanung geringfügig bessere Ergebnisse liefern als das einfache Verfahren, das wir simuliert haben.

Der größte Teil der Bewegung findet entlang einer Straße statt. Abbiegevorgänge sind verglichen mit der Bewegungsdauer entlang der Straße selten. Eine Straße wiederum besitzt nur selten eine Kurve, von der eine andere Straße gerade aus weiterführt. In diesem Fall hätten die beiden komplexeren Verfahren gegenüber dem einfachen Kartenverfahren einen Vorteil, weil sie die abbiegende Vorfahrtsstraße erkennen können. Das einfache Verfahren würde die Straße wählen, welche den geringsten Unterschied zur Momentanen Teilstrecke besitzt und damit eine falsche Auswahl treffen. Ein Update wäre nötig. (siehe Abbildung 39)

Durch Routenplanung hätte das Verfahren auch noch dann einen Vorteil, wenn von einer Straße in eine andere abgebogen werden muß, um weiter Richtung Ziel zu gelangen. Aber diese Abbiegevorgänge sind sehr selten verglichen mit der Länge einer Gesamtstrecke, so daß der Vorteil nur gering wäre.

Auch bei der Routenplanung oder beim wahrscheinlichkeitsbasierten Verfahren lassen sich Änderungen der Geschwindigkeit nicht vorhersagen. Ein Ampelstopp erfordert auch bei diesen Verfahren ein zusätzliches Update! Da der Großteil an Updates auf Geschwindigkeitsänderungen des Fahrzeugs zurückzuführen sind, dürfte der Vorteil der

beiden komplexeren Verfahren kaum ins Gewicht fallen.

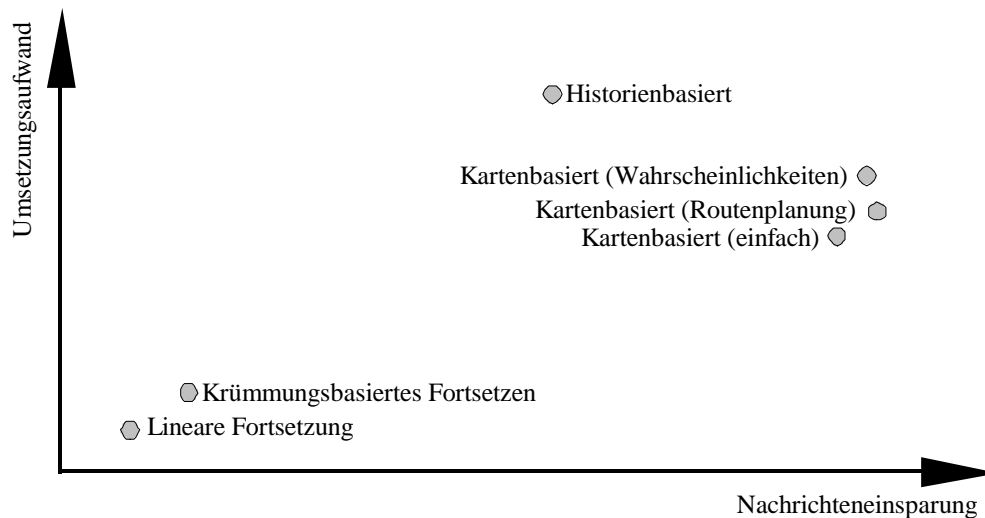


Abbildung 37 – Einordnung der verschiedenen Koppelnavigationsverfahren hinsichtlich der zu erwarteten Performance

Damit wurde begründet, daß die beiden hier untersuchten Verfahren das untere und das obere Ende der Skala bilden dürften, was die Einsparung an Updatenachrichten gegenüber einem Verfahren ohne Koppelnavigation angeht.

## 7.5.2 Mögliche Probleme

Nachfolgend werden ein paar mögliche Problemquellen aufgezeigt und diskutiert, welche die Ergebnisse beeinflussen können.

### 7.5.2.1 Meßsystem

Das von uns verwendete DGPS-Meßsystem hat bei einer Genauigkeit  $d_{max}$  von etwa 10m seine Grenzen. Daher ist es nicht möglich, ein Mobiles Objekt genauer zu lokalisieren. Ist dies gewünscht, muß ein Meßsystem eingesetzt werden, das die Positionen mit einer größeren Genauigkeit bestimmen kann.

Eine größere Genauigkeit ließe sich beispielsweise durch Verwendung von DGPS in Kombination mit Radsensoren realisieren, weil diese die zurückgelegten Strecken genau messen können.

### 7.5.2.2 Kommunikation

In der hier durchgeführten Simulation wurde die Anzahl der entstandenen Updatenachrichten gezählt, um ein Maß zum Vergleich der Verfahren untereinander zu gewährleisten. Unbeachtet blieb dagegen die Simulation des Kommunikationskanals zwischen Client und Server.

Besonders stark könnten sich die Eigenschaften des Kommunikationskanals auswirken, wenn

es zu langen Laufzeiten zwischen dem Versenden einer Nachricht und der Ankunft beim Partner kommt. Updates kommen langsam an und sind eventuell nicht mehr hinreichend aktuell. Außerdem ist die Erkennung von Verbindungsabbrüchen schwieriger.

Es kommt zu Retransmits und damit zu zusätzlicher Kommunikation, weil Timeouts ablaufen bevor die Antwortnachricht zurückkommt. Damit erhöht sich die benötigte Bandbreite zur Kommunikation, ohne daß mehr Updates verschickt wurden. Dadurch wäre die Kommunikation nicht mehr nur Abhängig vom eingesetzten Dead-Reckoning-Verfahren.

### **7.5.2.3 Vorgegebene Genauigkeit**

Die Verfahren wie sie hier betrachtet wurden, haben stets einen festen Wert für die maximale Ungenauigkeit  $d_{max}$ . Benötigt nun aber ein System eine höhere Genauigkeit für den Standort eines Mobilobjektes, ist der Server nicht in der Lage, diese Anfrage zu beantworten.

Das in Kapitel 4.6 (Seite 39) vorgestellte Kommunikationsprotokoll müßte so erweitert werden, daß es dem Server möglich wird, bei Bedarf explizit eine Anfrage an den Client zu schicken, um dessen Position mit einer größeren Genauigkeit zu erfragen.

Das bedeutet aber zusätzliche Kommunikation und damit eine zusätzliche Belastung des Kommunikationskanals. Zudem müßte das anfragende System eine gewisse Zeit warten bis seine Anfrage vom Server beantwortet werden kann, da der Server erst auf die Antwort des Clients warten muß.

Damit hängt die Performance der Verfahren auch zum Teil von der Art und Häufigkeit der Anfragen Dritter an den Server über den Aufenthaltsort eines Mobilobjektes ab. Nur wenn andere Systeme keine Standortnachfragen mit höheren Genauigkeiten stellen dürfen als die, die vom Server zur Verfügung gestellt werden, darf dieser Sachverhalt außer Acht gelassen werden.

## **7.5.3 Verbesserungen der Verfahren**

Neben der möglichen Fehlerquellen stellt sich auch die Frage, wie die erzielten Ergebnisse zusätzlich verbessert werden könnten. Nachfolgend werden dazu ein paar Vorschläge gemacht, wie dies erfolgen könnte. Eine Überprüfung der Vorschläge durch Simulation bleibt allerdings späteren Arbeiten vorbehalten.

### **7.5.3.1 Berücksichtigung der Beschleunigung**

In unserer Implementierung wird stets angenommen, daß sich die Objekte mit einer konstanten Geschwindigkeit weiterbewegen. Die Geschwindigkeitsinformation wird für die Berechnung der Soll-Positionen aus den Updatenachrichten abgelesen.

Denkbar wäre es aber nicht nur von einer gleichförmigen Bewegung auszugehen, sondern statt dessen Beschleunigungen zu berücksichtigen. Um Mißverständnissen vorzubeugen, sei klargestellt, daß immer wenn nachfolgend das Wort „Beschleunigung“ benutzt wird, sowohl positive (beim Beschleunigen) als auch negative (beim Bremsen) Beschleunigung gemeint ist.<sup>12</sup>

---

<sup>12</sup> Die Benutzung des Begriffes entspricht also der Bedeutung in der Mechanik und nicht dem umgangssprachlichen Gebrauch, bei dem nur die positive Beschleunigung vom Wort abgedeckt wird!

Der einfache Ansatz, sich den Geschwindigkeitsverlauf der letzten Sekunden anzusehen und daraus eine Beschleunigung  $a$  zu berechnen, führt aber zu keiner Verbesserung der Resultate des Verfahrens. Wenn man sich die Anzahl der Updates pro Kilometer ansieht, stellt man fest, daß sehr wenig Updates pro Streckeneinheit erfolgen. Selbst in der Stadt erreichen die Verfahren in der Regel Strecken, die viel größer sind als die Beschleunigungsstrecken.

Die Folge wäre, daß sich die Resultate verschlechtern. Positive Beschleunigungen – beispielsweise beim Anfahren nach einem Ampelhalt – würden dazu führen, daß sehr schnell Geschwindigkeiten angenommen werden, die weit über den realen Werten lägen. Das bedeutet, daß der Abstand zwischen den Soll-Positionen und der Ist-Positionen sehr schnell steigt. Zusätzliche Updates wären die Folge.

Negative Beschleunigungen wären dagegen unproblematisch, weil im Grenzfall angenommen wird, daß sich das Mobile Objekt gar nicht bewegt, also die Soll-Geschwindigkeit den Wert 0 erreicht. Damit wäre das Verfahren stets besser als ein Distance-Based Reporting Protokoll, bei dem die Geschwindigkeit stets als 0 angenommen wird.

Als Fazit kann man sagen, daß die Berücksichtigung der Beschleunigung für die gegebenen Verhältnisse nicht sinnvoll ist!

Eine eventuelle Verbesserung ist bei der Verwendung der Beschleunigung nur dann zu erwarten, wenn neben der Maßzahl der Beschleunigung noch Annahmen getroffen werden, welche Zielgeschwindigkeit (also Maximalgeschwindigkeit) angenommen werden soll. Erreicht die Soll-Geschwindigkeit die Zielgeschwindigkeit, so wird keine weitere Beschleunigung mehr eingerechnet. Anschließend wird die Zielgeschwindigkeit für Berechnungen weiterbenutzt. Das bedeutet also, daß zunächst eine Beschleunigung angenommen wird, die nach Erreichen der Zielgeschwindigkeit in eine konstante Geschwindigkeit übergeht.

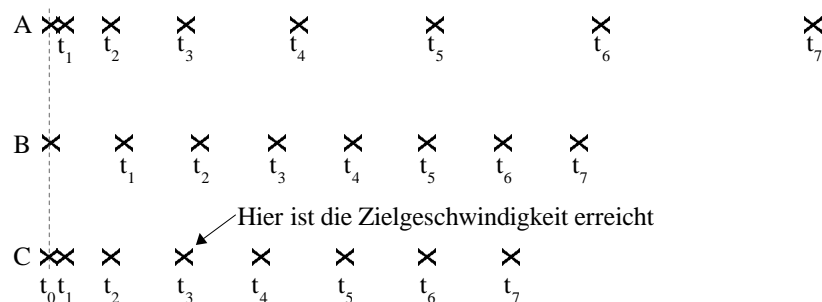


Abbildung 38 – Darstellung des Unterschieds der zurückgelegten Strecken bei Verwendung einer Beschleunigung (A), bei konstanter Geschwindigkeit (B) und bei Beschleunigung bis zu einer Zielgeschwindigkeit (C). Beachte: Bei B und C sind die (Ziel-)Geschwindigkeiten gleich. A und C haben die Beschleunigung gemeinsam.

Die Festlegung der Zielgeschwindigkeit hängt von dem verwendeten Dead-Reckoning-Algorithmus und dem Einsatzzweck ab. Vorstellbar wäre eine ganze Reihe von Festlegungen. Man könnte Maximal- oder Durchschnittsgeschwindigkeiten festlegen. Diese könnten wiederum global, lokal oder klassenbasiert sein (oder eine Mischung). Unter einer globalen Festlegung ist zu verstehen, daß die Geschwindigkeit für alle Mobilen Objekte unabhängig von ihrem Aufenthaltsort gilt. Dagegen bezeichnet lokal, daß der jeweilige Wert je nach

Aufenthaltort variiert. Klassenbasiert bedeutet, daß für jede Klasse von Mobilien Objekten (z.B. PKW, LKW, Fußgänger, etc.) eine eigene Geschwindigkeit festgelegt wird.

Die Zielgeschwindigkeiten könnten entweder Client und Server vorher anhand einer (jeweils eigenen) Datenbank bekannt sein, beim Verbindungsaufbau zwischen den beiden ausgehandelt werden oder dynamisch mit jeder Updatenachricht übertragen werden.

Am einfachsten lassen sich die Zielgeschwindigkeitsinformationen beim Kartenbasierten Verfahren einbauen. Bei diesem Verfahren müssen Client und Server bereits über eine Karte verfügen. Durch hinzufügen eines einzelnen Attributes zu einer Kartenkante, könnte man die Zielgeschwindigkeit einbauen. Damit wäre bereits ein sehr anpaßbarer und flexibler Ansatz realisiert, weil man für jede Kante einen eigenen Wert vergeben kann.

### **7.5.3.2 Map Matching**

In dieser Arbeit wurde ein recht einfaches Map Matching Verfahren eingesetzt. Trotzdem ist die Verbesserung gegenüber dem Verfahren des Linearen Fortsetzens deutlich an den Ergebnissen ablesbar. Nachfolgend werden ein paar Ansatzpunkte präsentiert, die zu einer weiteren Verbesserung der Resultate führen.

Das sind im einzelnen ein komplexeres Map Matching Verfahren, um die Auswahlssicherheit einer Kante zu verbessern, Verbesserungen bei den Kartendaten, um eine Unterscheidung nach Haupt- und Nebenstraßen zu ermöglichen und die Berücksichtigung von Verkehrszeichen.

#### **Besserer Kartenabgleich**

Die Ergebnisse ließen sich weiter steigern, wenn statt des einfachen Verfahrens ein komplexeres Verfahren für den Kartenabgleich herangezogen werden würde, da gelegentlich der hier verwendete Kartenabgleich falsche Kanten aussucht. Liefert der Kartenabgleich zuverlässigere Resultate, werden auch Updatenachrichten eingespart, was bedeuten würde, daß das Verfahren noch bessere Ergebnisse erzielt.

Entscheidet sich das Map-Matching-Verfahren für eine falsche Kartenkante, bedeutet dies, daß falsche Soll-Positionen berechnet werden. Das hat zur Folge, daß unnötige Updates verschickt werden. Wird die richtige Kante ausgesucht – was schließlich das Ziel des Verfahrens ist – werden richtige Soll-Positionen berechnet. Im Gegensatz zu den falschen Soll-Positionen, ist bei den richtigen Soll-Lagen, die Wahrscheinlichkeit höher, daß die Ist-Positionen besser vorhergesagt werden können, weil nicht bereits falsche Annahmen der Berechnung zugrunde liegen.

#### **Benutzung der Straßen-Hierarchie**

Ein weiterer Punkt ist der, daß die Implementierung keinen Unterschied macht zwischen einer Hauptstraße und einer Nebenstraße. Alle Kanten der Karte sind gleichberechtigt. Nach einem Knotendurchlauf wird für die Berechnung der Soll-Positionen stets angenommen, daß diejenige Kante vom Mobilien Objekt genommen wird, welche zur bisherigen Kante, den kleinsten Richtungsunterschied aufweist.

Es ist zu erwarten, daß sich die Ergebnisse nochmals verbessern, wenn nach einem

Knotendurchlauf die Kante ausgewählt wird, die der bisherigen Straße entspricht. Dadurch würden auch abbiegende Hauptstraßen vom Algorithmus richtig behandelt werden. Die Begründung ist die, daß die Wahrscheinlichkeit für den Verbleib eines Mobilobjektes auf einer Straße viel höher ist als für ein Abbiegen auf eine andere Straße. (Anschaulich begründet: Abbiegevorgänge sind viel seltener als das Vorbeifahren an Kreuzungen.)

Das Kartenbasierte Verfahren mit Wahrscheinlichkeiten hätte die Hierarchie der Straßen implizit in den Wahrscheinlichkeiten gespeichert. Da die meisten Objekte auf der gleichen Straße verbleiben, besitzt die Kante, welche die Straße fortführt, einen höheren Wahrscheinlichkeitswert. Diese Kante würde nach einem Knotendurchlauf ausgewählt werden.

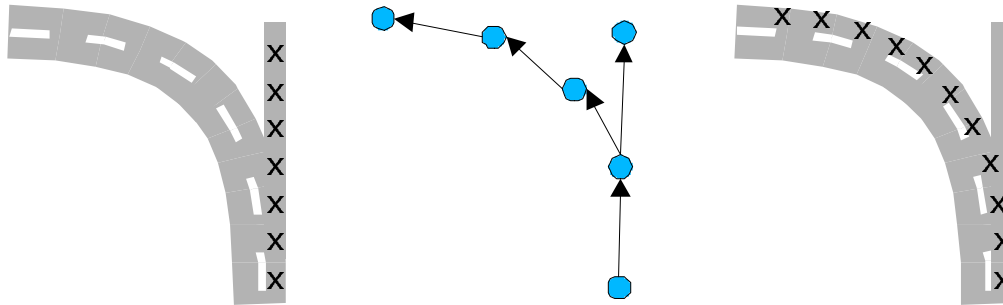


Abbildung 39 – Links sind die berechneten Soll-Positionen eingezeichnet, wenn die Kante mit der geringsten Krümmung zur Fortsetzung genommen wird. Rechts dagegen, wird angenommen, daß sich das Objekt entlang der gleichen Straße weiterbewegt. In der Mitte ist die Repräsentation der Kreuzung durch Kanten und Knoten dargestellt.

## Verkehrsschilder

Eine weitere Möglichkeit zur Verbesserung der Ergebnisse wäre die Einbeziehung von Verkehrszeichen in die Karte. Speziell Schilder für Geschwindigkeitsbegrenzungen und Abbiegegebote/-verbote ließen sich für das Verfahren einsetzen.

Durch die Geschwindigkeitsbegrenzungen ließen sich auf sehr einfache Weise Zielgeschwindigkeit realisieren wie sie weiter oben für die Verwendung von Beschleunigungen vorgeschlagen wurden.

Durch die Interpretation von Abbiegeschildern könnte die Zahl der von einem Knoten ausgehenden Kanten reduziert werden. Das kann helfen, falsches Matching zu einer Kante zu vermeiden.

## 8 Fazit und Zusammenfassung

---

Das Ziel der Arbeit war es, Dead-Reckoning-Verfahren aufzustellen und zu bewerten. Dabei sollte durch den Einsatz der Verfahren, Kommunikationsbandbreite zwischen dem Lokationsserver und dem Client eingespart werden. Trotzdem sollte der Server stets mit einer vorgegebenen maximalen Ungenauigkeit  $d_{max}$  wissen, wo sich ein Mobiles Objekt befindet.

Es wurden diverse Koppelnavigations-Verfahren eingeführt und deren Realisierung besprochen. Manche davon sind mit geringerem Aufwand verbunden, wie die Fortsetzungsverfahren, andere dagegen benötigen mehr CPU-Leistung und zusätzliche Daten wie die kartenbasierten Verfahren. Man hat sehen können, daß der Großteil der Implementierung des Ablaufrahmens und der Kommunikation sowohl für den Client als auch für den Server unabhängig vom verwendeten Algorithmus gleich ist. Durch Ausnutzung des Polymorphismus und des Dynamischen Bindens wird die Umsetzung eines neuen Verfahrens erleichtert, wenn das Framework einmal steht.

Wir haben zwei ausgewählte Verfahren, das Lineare Fortsetzen und das Map Matching Verfahren, genauer betrachtet und zur Implementierung hingeführt. Diese beiden Verfahren bildeten den Grundstock für die Simulation. Das Lineare Fortsetzen wurde wegen seiner Einfachheit ausgesucht, gegen welches das kartenbasierte Verfahren antreten sollte. Vom linearen Verfahren wurde erwartet, daß es wegen seiner Simplizität von allen Dead-Reckoning-Verfahren die geringsten Einsparungen an Updatenachrichten gegenüber einem Nicht-Dead-Reckoning-Protokoll liefert. Dagegen sollte das kartenbasierte Verfahren wegen seiner Komplexität, die besten Resultate bringen.

In der Simulation bestätigten sich diese Erwartungen qualitativ. Quantitativ überraschte das Lineare Fortsetzen – trotz seiner Einfachheit – bereits mit sehr starken Verbesserungen gegenüber einem Nicht-Dead-Reckoning-Verfahren. Sie lagen in der Größenordnung von 83% bis 55% außerorts. Bei Stadtfahrten konnten dagegen „nur“ 63% bis 18% der Nachrichten eingespart werden. Zu Fuß sind bis zu 67% erreichbar. Das Kartenverfahren brachte zudem noch eine weitere Verbesserung mit sich. Es benötigte nur noch 40% bis 95% der Updateraten des Linearen Verfahrens. Das entspricht gegenüber einem Nicht-Dead-Reckoning-Verfahren, daß nur 9% bis 63% der Updates verschickt werden müssen! Selbst im schlechtesten Fall entspricht dies einer Einsparung von einem Drittel der Nachrichten!

Insgesamt zeigten die Dead-Reckoning-Protokolle überraschend hohe Einsparmöglichkeiten. Eine Umsetzung in die Praxis für Location Based Services lohnt sich daher. Bei geringen verfügbaren Hardware-Ressourcen beim Client, läßt sich bereits das Lineare Fortsetzen implementieren, das schon eine große Reduktion der Kommunikationsbandbreite mit sich bringt! Stehen mehr Ressourcen zur Verfügung, kann der Kartenalgorithmus umgesetzt werden, um noch bessere Ergebnisse zu erzielen.

Für eine Umsetzung der beiden Verfahren, müßte man die in der vorliegenden Arbeit angestellten Überlegungen hinsichtlich des Kommunikationsprotokolls und der Abläufe

(insbesondere auf Server-Seite) detaillieren und konkretisieren. Der Rahmen ist aber bereits vorgezeichnet.

Mit ein paar Verbesserungen beim Map Matching und bei den Kartendaten, ließen sich die Ergebnisse des kartenbasierten Verfahrens zusätzlich etwas noch optimieren, was zwar das Verfahren bzw. dessen Daten komplexer macht, aber dafür die Ergebnisse verbessert.

Zusammenfassend läßt sich sagen, daß sich der Aufwand für die Umsetzung von Koppelnavigations-Protokollen zur Verfolgung der Aufenthaltsorte von Mobilien Objekten sehr lohnend erscheint. Eine Umsetzung ist mit der zur Zeit verfügbaren Technik (Mobilfunknetze oder Wireless LANs) bereits möglich!

## 9 Begriffslexikon

---

Im folgenden Begriffslexikon werden wichtige Begriffe definiert und erläutert. Dies ist insbesondere für solche Begriffe wichtig, die im Kontext der Aufgabenstellung eine besondere Bedeutung haben oder neu eingeführt werden. Oft ist es aber auch nötig, die Bedeutung eines Begriffs zu definieren, der allgemein bekannt ist, aber unterschiedliche Bedeutungsnuancen besitzt, um Mißverständnisse zu vermeiden.

Die Zusammenstellung soll dem Leser das Verständnis erleichtern und als Hilfe dienen, wenn in einem Kapitel ein Begriff erneut gebraucht wird, der aber an einer früheren Stelle erklärt wurde und der Leser die Bedeutung nachschlagen möchte.

<i>Begriff</i>	<i>Erklärung/Definition</i>
Adaptive Arithmetik	Selbstgeschriebene Software-Module, die auf eine einheitliche Art Berechnungen durchführen. Sie dienen dazu, um sicherzustellen, daß die Resultate von Berechnungen auf unterschiedlichen Prozessoren gleiche Ergebnisse liefert.
Backtracking	Verfahren zum Finden der richtigen Kartenkante, wenn sich das Map Matching als falsch herausgestellt hat. (siehe Kapitel 6.2.4.2, Seite 70)
Client	Wird hier als Synonym für Mobiles Objekt (siehe dort) verwendet.
Datenbank	Unabhängig davon wird im folgenden der Begriff „Datenbank“ verwendet, um das Datenhaltungssystem zu bezeichnen.
Dead-Reckoning	englischer Begriff für „Koppelnavigation“.
Forwardtracking	Name der Vorgehensweise, das beim kartenbasierten Verfahren, eine Folgekante aussucht, wenn das Mobile Objekt eine Kante verlassen hat. (siehe Kapitel 6.2.4.2, Seite 69)
Ist-Position	Die Position, an der sich das mobile Objekt befindet. Diese Position wird bestimmt vom einem Lagemeßsystem (z.B. GPS-Empfänger). Daher ist die Genauigkeit der Ist-Position abhängig von der Genauigkeit des zugrundeliegenden Lagemeßsystems.

<i>Begriff</i>	<i>Erklärung/Definition</i>
Koppelnavigationsprotokoll	Bezeichnung für ein Protokoll, bei dem ausgehend von einer bekannten Position alle weiteren Positionen auf Grund von Vorgaben wie Bewegungsrichtung und Geschwindigkeit von einem Algorithmus berechnet werden. Die berechneten Positionen heißen Soll-Positionen.
Lageupdate	Nachricht mit Lage- und Bewegungsinformationen, die vom mobilen Objekt an den Lokationsserver geschickt wird.  Das Lageupdate wird immer dann verschickt, wenn die Soll-Lage von der Ist-Lage mehr als ein festgelegtes Maximum abweicht. Das jeweils zuletzt empfangene Lageupdate dient als Ausgangspunkt für die Berechnung der Soll-Position durch den Dead-Reckoning-Algorithmus.
Lineares Fortsetzen	Ein Koppelnavigationsverfahren, das davon ausgeht, daß sich das Mobile Objekt mit der gleichen Geschwindigkeit und der gleichen Richtung weiterbewegen wird wie bisher.
Link	Kante zwischen zwei Knoten in der elektronischen Karte
Lokationsserver	Der Lokationsserver steht hier für das System, das die Lokationen der mobilen Objekte speichert bzw. die aktuellen Soll-Positionen errechnet.  Zum einen muß der Lokationsserver Update-Nachrichten seitens der mobilen Objekte verarbeiten. Zum anderen ist der Lokationsserver dazu da, Anfragen zu den aktuellen Positionen der Objekte zu beantworten
Lotfußpunkt	Der Punkt $P^*$ auf einer Geraden $g$ , der von einem Punkt $P$ den minimalen Abstand besitzt.  $P^*$ liegt zwangsläufig im Schnittpunkt der Geraden $g$ und der Geraden $g^*$ , die orthogonal zur Geraden $g$ ist und durch den Punkt $P$ geht.
Map Matching	Abgleich einer gemessenen Position mit der zugrundeliegenden Karte. Der Grundgedanke ist, daß ein mobiles Objekt nur auf den durch die Karte vorgegebenen Kanten sein kann. Dazu wird die gemessene Lage auf die (anhand der vorliegenden Daten) bestimmte Karten-Kante projiziert.
Matching	Siehe Map Matching
Mobiles Objekt	Dieser Begriff bezeichnet hier das Objekt (z.B. Auto oder Fußgänger), das nicht stationär ist (daher mobil) und dessen Position (mit einer zugesicherten maximalen Abweichung) zu jedem Zeitpunkt von Lageserver erfragt werden kann.
Node	Synonym für „Knoten“ in einer elektronischen Karte

<i><b>Begriff</b></i>	<i><b>Erklärung/Definition</b></i>
Orientierung einer Kante	<p>Die Orientierung einer Kante entspricht der geographischen Richtung, in die diese Kante geht. Die Orientierung wird durch einen Winkel in Grad ausgedrückt.</p> <p>Dabei gelten hier folgende Werte:</p> <p><math>\alpha=0^\circ</math> entspricht Nord  <math>\alpha=90^\circ</math> entspricht West  <math>\alpha=180^\circ</math> entspricht Süd  <math>\alpha=270^\circ</math> entspricht Ost</p>
Ping-Nachricht	Unter einer Ping-Nachricht ist eine Nachricht zu verstehen, die vom Datenvolumen sehr klein ist und keine Informationen übermittelt. Sie dient lediglich dazu, dem Server zu zeigen, daß die Verbindung zwischen Client und Server noch besteht.
Server	Steht hier als Synonym für Lokationsserver (siehe dort)
Soll-Position	Die Position, von der angenommen wird, daß sich das mobile Objekt an dieser Stelle befindet. Diese Lage wird durch das Dead-Reckoning-Protokoll unter Zuhilfenahme des letzten Lageupdates errechnet.
Toleranzradius	Bezeichnung für den maximal erlaubten Abstand zwischen der gemessenen Ist-Position und der berechneten Soll-Position. Wird dieser Abstand überschritten, sendet der Client eine Updatenachricht an den Server.
Toleranzschwelle	als Synonym für Toleranzradius verwendet. Siehe Toleranzradius

# 10 Quellenverzeichnis

---

- [BAU97] Vermessung und Ortung mit Satelliten, M. Bauer, Wichman-Verlag, 1997
- [BER99] Krümmungsbasiertes Map Matching – ein Verfahren zur Positionsbestimmung, Britta Bernecker, Diplomarbeit am Institut für Photogrammetrie, 1999
- [CZO00] Renate Czommer, Leistungsfähigkeit fahrzeugautonomer Ortungsverfahren auf der Basis von Map-Matching-Techniken, Institut für Anwendungen der Geodäsie, Universität Stuttgart, Dissertation 2000
- [D2MAN] Homepage Netzbetreiber D2, [www.d2mannesmann.de](http://www.d2mannesmann.de)
- [FRÖ95] Punktbestimmung mit GPS für Einsteiger, H. Fröhlich, S. Grimm, Dümmer-Verlag, 1995
- [GDF95] Geographical Data File, CEN (Europäisches Komitee für Normung), Version 3.0, 1995
- [HAY88] Computer Architecture And Organization, John P. Hayes, McGraw-Hill, 1998
- [HEC95] Rechenverfahren und Auswertungsmodelle der Landesvermessung, B. Heck, Wichman-Verlag, 1995
- [IEE85] IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985), IEEE Inc., New York, August 1985
- [JAVA] Spezifikation der Virtual Machine, SUN Microsystems, [www.javasoft.com](http://www.javasoft.com)
- [KAO94] Kao, W.-W. und L. J. Huang, System and Method for Locating a Traveling Vehicle, US Patent No. 5,283,575, February 1994
- [LEO01] Alexander Leonhardi; Kurt Rothermel, A Comparison of Portocols for Updating Location Information, To appear in Special Issue on Spatial Location in Networking, Baltzer Cluster Computing Journal, Baltzer Science Publishers, 2001.
- [MERZ94] Formeln + Hilfen zur höheren Mathematik, Gerhard Merziger, Günter Mühlbach, Detlev Wille, Thomas Wirth, Binomi-Verlag, 1995
- [NAV] Homepage NavTech, [www.navtech.com](http://www.navtech.com)

- [NEXUS] Projekthomepage NEXUS-Projekt, [www.nexus.uni-stuttgart.de](http://www.nexus.uni-stuttgart.de)
- [PIL98] Roman Pils, Vergleich von Map Matching Algorithmen, Institut für Anwendungen der Geodaesie im Bauwesen, Universität Stuttgart, Diplomarbeit 1998
- [RAB99] Dietmar Rabel, Map-Matching zur Optimierung der Positionsbestimmung von Fahrzeugen, Institut für Navigation, Universität Stuttgart, Diplomarbeit 1999
- [RIV92] R.L. Rivest, The MD5 Message-Digest Algorithm, RFC 1320, 1992
- [TMOT] Homepage T-Motion, [www.t-motion.de](http://www.t-motion.de)
- [VIAG] Homepage Viag-Interkom, [www.viag-interkom.de](http://www.viag-interkom.de)
- [WOL99] Updating and Querying Databases that Track Mobile Units, Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, Yelena Yesha, Distributed and Parallel databases Journal, 7(3):1-31, 1999

# Anhang

## Einzelne Ergebnisse

Die folgenden Tabellen zeigen die absolute Anzahl an Updatenachrichten. Über einer Tabelle ist jeweils der Toleranzradius  $d_{max}$  (in Metern) und die Anzahl der Punkte (*Regressionspunkte*) für die Richtungsbestimmung angegeben. Die  $m_{max}$ -Werte stehen für verschiedene Werte in Metern.

Unter den  $m_{max}$ -Werte stehen die Resultate für den Kartenalgorithmus. Die Bezeichnung „lin.DR“ steht für das Lineare Fortsetzen und „no DR“ entspricht dem Distance-Based-Reporting-Protokoll.

### $d_{max}=20$ , 4 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	60	61	124	226	329	425	497	560	97	736
	Autobahn_2.txt	63	49	141	307	426	522	625	714	110	1122
	Autobahn_3.txt	50	50	81	103	132	154	191	238	48	408
	Autobahn_4.txt	59	54	76	101	139	170	233	257	93	363
	Autobahn_5.txt	46	40	54	74	113	141	202	222	82	322
	Autobahn_6.txt	177	171	187	215	234	248	271	281	205	379
	Autobahn_7.txt	162	155	169	181	181	194	224	239	185	342
	Summe:	617	<b>580</b>	832	1207	1554	1854	2243	2511	820	3672
Ueberland	Ueberland_1.txt	328	331	490	864	1043	1137	1278	1375	378	1801
	Ueberland_2.txt	90	93	84	111	166	206	224	253	103	314
	Ueberland_3.txt	46	42	78	97	124	157	183	181	67	254
	Ueberland_4.txt	104	108	109	136	162	188	221	232	112	290
	Ueberland_5.txt	180	187	210	239	257	250	277	294	232	416
	Summe:	<b>748</b>	761	971	1447	1752	1938	2183	2335	892	3075
Stadt	Stadt_1.txt	64	63	80	120	133	145	161	167	77	202
	Stadt_3.txt	126	145	194	213	264	280	288	313	142	461
	Stadt_4.txt	216	263	305	387	449	468	522	507	270	663
	Stadt_5.txt	186	202	256	291	317	368	376	380	188	561
	Stadt_6.txt	93	67	78	119	135	140	158	186	79	223
	Stadt_7.txt	172	179	197	214	194	187	219	234	124	286
	Stadt_8.txt	117	133	151	171	187	207	243	262	138	346
		Summe:	<b>974</b>	1052	1261	1515	1679	1795	1967	2049	1018
Fussgaenger	Fussgaenger_1.txt	16	24	37	52	58	62	60	61	23	76
	Fussgaenger_2.txt	92	72	86	94	94	102	127	137	57	125
	Fussgaenger_3.txt	38	23	25	42	46	48	57	54	24	67
	Fussgaenger_4.txt	60	40	82	101	131	141	154	158	45	166
	Summe:	206	<b>159</b>	230	289	329	353	398	410	149	434

## $d_{\max}=20$ , 8 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	65	61	117	244	330	426	497	562	125	736
	Autobahn_2.txt	61	54	123	298	434	525	626	714	132	1122
	Autobahn_3.txt	57	57	85	106	127	152	196	239	55	408
	Autobahn_4.txt	65	67	69	95	130	169	235	261	124	363
	Autobahn_5.txt	50	40	63	83	113	136	199	224	115	322
	Autobahn_6.txt	178	171	183	207	231	243	274	283	236	379
	Autobahn_7.txt	168	166	180	187	190	199	225	240	196	342
	<i>Summe:</i>	<b>644</b>	<b>616</b>	820	1220	1555	1850	2252	2523	983	3672
Ueberland	Ueberland_1.txt	378	354	520	855	1019	1126	1272	1374	483	1801
	Ueberland_2.txt	100	100	102	129	180	204	237	256	123	314
	Ueberland_3.txt	56	58	77	104	132	167	186	193	93	254
	Ueberland_4.txt	103	98	100	126	153	186	216	229	127	290
	Ueberland_5.txt	155	181	222	239	268	262	280	290	239	416
	<i>Summe:</i>	792	<b>791</b>	1021	1453	1752	1945	2191	2342	1065	3075
Stadt	Stadt_1.txt	67	70	83	113	136	145	161	167	88	202
	Stadt_3.txt	145	146	190	223	264	279	299	312	180	461
	Stadt_4.txt	255	283	332	406	467	483	532	522	316	663
	Stadt_5.txt	191	209	279	311	340	375	408	392	223	561
	Stadt_6.txt	106	80	89	120	136	143	162	190	102	223
	Stadt_7.txt	169	163	203	200	202	201	222	243	150	286
	Stadt_8.txt	127	130	155	191	192	204	254	268	156	346
		<i>Summe:</i>	<b>1060</b>	1081	1331	1564	1737	1830	2038	2094	1215
Fussgaenger	Fussgaenger_1.txt	21	21	28	49	58	59	55	59	21	76
	Fussgaenger_2.txt	86	65	73	91	94	115	125	141	50	125
	Fussgaenger_3.txt	32	20	33	38	48	53	54	57	20	67
	Fussgaenger_4.txt	48	43	70	115	139	140	158	165	53	166
	<i>Summe:</i>	187	<b>149</b>	204	293	339	367	392	422	144	434

$d_{\max}=50$ , 4 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	28	27	33	31	33	79	137	203	59	361
	Autobahn_2.txt	34	25	26	29	33	73	261	256	66	554
	Autobahn_3.txt	28	30	26	21	22	35	100	102	25	197
	Autobahn_4.txt	39	33	25	28	34	35	95	154	52	339
	Autobahn_5.txt	27	20	23	27	27	31	81	108	49	309
	Autobahn_6.txt	49	51	54	79	73	81	117	139	90	361
	Autobahn_7.txt	37	32	37	47	60	65	83	121	68	327
	Summe:	242	<b>218</b>	224	262	282	399	874	1083	409	2448
Ueberland	Ueberland_1.txt	160	154	144	166	168	245	397	422	217	781
	Ueberland_2.txt	42	33	34	34	35	51	110	118	53	159
	Ueberland_3.txt	25	21	20	27	24	39	63	64	36	131
	Ueberland_4.txt	43	41	37	46	41	59	93	112	51	185
	Ueberland_5.txt	43	43	53	63	87	97	113	131	69	237
	Summe:	313	292	<b>288</b>	336	355	491	776	847	426	1493
Stadt	Stadt_1.txt	34	31	31	31	37	41	62	73	42	101
	Stadt_3.txt	73	73	63	63	82	84	107	135	68	220
	Stadt_4.txt	111	111	112	129	149	182	214	220	134	311
	Stadt_5.txt	89	89	88	93	108	124	170	186	96	266
	Stadt_6.txt	52	36	40	39	41	50	64	94	47	99
	Stadt_7.txt	93	93	97	96	81	73	97	120	64	140
	Stadt_8.txt	55	57	61	62	67	79	116	114	61	161
		Summe:	507	<b>490</b>	492	513	565	633	830	942	512
Fussgaenger	Fussgaenger_1.txt	11	9	12	10	11	18	21	27	15	31
	Fussgaenger_2.txt	45	34	40	33	29	39	66	52	25	51
	Fussgaenger_3.txt	17	10	9	7	13	14	17	22	13	26
	Fussgaenger_4.txt	34	18	21	20	33	40	36	56	23	66
	Summe:	107	71	82	<b>70</b>	86	111	140	157	76	174

## $d_{\max}=50$ , 8 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	34	34	33	32	38	73	146	202	66	361
	Autobahn_2.txt	34	28	28	31	30	64	235	254	72	554
	Autobahn_3.txt	29	29	29	24	23	29	71	93	31	197
	Autobahn_4.txt	35	32	29	29	33	44	98	144	62	339
	Autobahn_5.txt	31	22	23	26	33	45	96	115	63	309
	Autobahn_6.txt	46	45	54	77	75	79	105	128	103	361
	Autobahn_7.txt	38	39	37	43	50	57	100	128	68	327
	<i>Summe:</i>	<b>247</b>	<b>229</b>	233	262	282	391	851	1064	465	2448
Ueberland	Ueberland_1.txt	179	170	166	186	201	249	403	440	241	781
	Ueberland_2.txt	54	46	47	45	52	61	92	112	61	159
	Ueberland_3.txt	32	29	33	33	29	36	73	67	46	131
	Ueberland_4.txt	48	39	38	41	43	55	70	101	63	185
	Ueberland_5.txt	60	56	63	67	97	100	116	141	93	237
	<i>Summe:</i>	373	<b>340</b>	347	372	422	501	754	861	504	1493
Stadt	Stadt_1.txt	37	33	34	38	39	41	65	77	45	101
	Stadt_3.txt	74	60	68	76	81	87	129	135	91	220
	Stadt_4.txt	120	111	113	133	156	182	228	247	150	311
	Stadt_5.txt	107	103	113	114	126	143	187	182	114	266
	Stadt_6.txt	55	45	46	42	45	48	63	98	58	99
	Stadt_7.txt	84	87	93	94	89	78	103	116	67	140
	Stadt_8.txt	59	57	59	62	68	80	105	127	80	161
		<i>Summe:</i>	536	<b>496</b>	526	559	604	659	880	982	605
Fussgaenger	Fussgaenger_1.txt	9	8	10	8	12	22	19	22	11	31
	Fussgaenger_2.txt	47	39	35	32	25	51	51	61	25	51
	Fussgaenger_3.txt	13	8	9	7	15	14	18	20	13	26
	Fussgaenger_4.txt	23	19	14	21	25	29	37	57	23	66
	<i>Summe:</i>	92	74	<b>68</b>	<b>68</b>	77	116	125	160	72	174

## $d_{\max}=100$ , 4 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	18	18	18	18	18	20	19	26	34	224
	Autobahn_2.txt	22	17	18	17	16	19	18	26	42	300
	Autobahn_3.txt	20	19	19	13	16	12	12	15	17	109
	Autobahn_4.txt	19	17	15	13	17	15	20	23	31	173
	Autobahn_5.txt	18	11	14	13	13	10	15	16	30	158
	Autobahn_6.txt	32	29	27	26	18	25	33	36	51	184
	Autobahn_7.txt	14	19	19	19	21	19	26	36	33	167
	<i>Summe:</i>		<b>143</b>	<b>130</b>	<b>130</b>	<b>119</b>	<b>119</b>	<b>120</b>	<b>143</b>	<b>178</b>	<b>238</b>
Ueberland	Ueberland_1.txt	106	98	88	84	88	97	119	122	139	429
	Ueberland_2.txt	26	23	21	22	20	28	28	35	32	97
	Ueberland_3.txt	16	14	13	14	14	15	17	16	24	79
	Ueberland_4.txt	27	23	21	22	23	27	34	32	35	100
	Ueberland_5.txt	32	28	31	27	40	37	46	58	43	141
	<i>Summe:</i>		<b>207</b>	<b>186</b>	<b>174</b>	<b>169</b>	<b>185</b>	<b>204</b>	<b>244</b>	<b>263</b>	<b>273</b>
Stadt	Stadt_1.txt	26	26	25	20	20	21	25	28	27	56
	Stadt_3.txt	32	30	33	33	36	34	34	43	44	116
	Stadt_4.txt	61	57	65	64	79	81	93	90	85	159
	Stadt_5.txt	53	58	48	54	56	56	72	74	62	137
	Stadt_6.txt	25	24	24	23	25	23	27	44	32	55
	Stadt_7.txt	38	33	34	34	33	34	50	61	37	75
	Stadt_8.txt	34	33	32	29	38	39	46	46	40	85
	<i>Summe:</i>		<b>269</b>	<b>261</b>	<b>261</b>	<b>257</b>	<b>287</b>	<b>288</b>	<b>347</b>	<b>386</b>	<b>327</b>
Fussgaenger	Fussgaenger_1.txt	6	7	5	6	6	6	8	9	7	15
	Fussgaenger_2.txt	17	17	18	15	19	22	21	22	12	25
	Fussgaenger_3.txt	11	4	4	5	5	6	7	7	7	13
	Fussgaenger_4.txt	10	6	8	8	10	12	17	30	17	32
	<i>Summe:</i>		<b>44</b>	<b>34</b>	<b>35</b>	<b>34</b>	<b>40</b>	<b>46</b>	<b>53</b>	<b>68</b>	<b>43</b>

## $d_{\max}=100$ , 8 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	18	18	18	18	21	21	19	22	44	224
	Autobahn_2.txt	24	16	15	13	14	17	18	23	46	300
	Autobahn_3.txt	19	18	18	15	12	11	13	13	16	109
	Autobahn_4.txt	23	19	15	15	15	16	18	22	38	173
	Autobahn_5.txt	22	10	10	10	10	11	17	21	37	158
	Autobahn_6.txt	26	21	22	27	20	19	28	38	56	184
	Autobahn_7.txt	11	13	12	12	12	16	28	31	37	167
	<i>Summe:</i>		143	115	110	110	<b>104</b>	111	141	170	274
Ueberland	Ueberland_1.txt	104	109	98	101	101	104	102	128	142	429
	Ueberland_2.txt	33	28	28	29	28	32	29	32	38	97
	Ueberland_3.txt	18	18	19	19	18	17	20	22	26	79
	Ueberland_4.txt	22	22	20	20	23	23	27	31	36	100
	Ueberland_5.txt	41	38	37	34	45	45	48	52	45	141
	<i>Summe:</i>		218	215	<b>202</b>	203	215	221	226	265	287
Stadt	Stadt_1.txt	23	24	22	21	20	21	27	29	26	56
	Stadt_3.txt	36	40	40	34	39	43	43	44	53	116
	Stadt_4.txt	67	66	64	68	75	80	83	92	90	159
	Stadt_5.txt	61	60	65	60	65	71	81	76	71	137
	Stadt_6.txt	31	27	27	27	30	26	30	48	32	55
	Stadt_7.txt	53	45	49	47	48	37	49	60	40	75
	Stadt_8.txt	37	38	41	37	33	37	45	55	48	85
	<i>Summe:</i>		308	300	308	<b>294</b>	310	315	358	404	360
Fussgaenger	Fussgaenger_1.txt	4	4	5	6	4	6	8	8	6	15
	Fussgaenger_2.txt	20	19	14	17	12	19	18	21	15	25
	Fussgaenger_3.txt	10	3	3	4	6	5	7	9	7	13
	Fussgaenger_4.txt	9	10	8	10	9	12	14	25	14	32
	<i>Summe:</i>		43	36	<b>30</b>	37	31	42	47	63	42

## $d_{\max}=150$ , 8 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Fussgaenger	Fussgaenger_1.txt	4	4	4	4	4	4	5	8	6	10
	Fussgaenger_2.txt	10	10	11	11	12	7	11	11	12	16
	Fussgaenger_3.txt	3	3	4	4	5	5	4	5	5	7
	Fussgaenger_4.txt	6	6	6	6	7	5	8	11	15	21
	<i>Summe:</i>		23	23	25	25	28	<b>21</b>	28	35	38

## $d_{\max}=200$ , 4 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	13	10	10	10	11	12	10	11	27	124
	Autobahn_2.txt	10	10	8	9	6	8	8	7	27	156
	Autobahn_3.txt	11	10	10	5	5	4	9	8	9	57
	Autobahn_4.txt	14	13	10	8	11	11	9	9	21	94
	Autobahn_5.txt	8	8	8	8	9	9	11	8	17	90
	Autobahn_6.txt	13	13	13	14	13	11	12	14	34	112
	Autobahn_7.txt	6	10	11	11	8	8	12	15	20	89
	<i>Summe:</i>	<i>75</i>	<i>74</i>	<i>70</i>	<i>65</i>	<b>63</b>	<b>63</b>	<i>71</i>	<i>72</i>	<i>155</i>	<i>722</i>
Ueberland	Ueberland_1.txt	55	56	52	54	54	57	61	58	88	220
	Ueberland_2.txt	15	14	15	14	14	16	15	16	20	53
	Ueberland_3.txt	9	12	10	6	7	7	10	7	14	44
	Ueberland_4.txt	15	14	15	15	15	15	15	16	22	56
	Ueberland_5.txt	17	16	15	16	25	23	25	22	26	75
	<i>Summe:</i>	<i>111</i>	<i>112</i>	<i>107</i>	<b>105</b>	<i>115</i>	<i>118</i>	<i>126</i>	<i>119</i>	<i>170</i>	<i>448</i>
Stadt	Stadt_1.txt	9	11	11	10	12	11	14	15	15	30
	Stadt_3.txt	22	21	22	20	21	20	21	22	30	59
	Stadt_4.txt	38	41	42	39	41	41	43	43	49	79
	Stadt_5.txt	35	37	30	34	35	39	36	36	38	68
	Stadt_6.txt	16	16	16	16	12	16	15	16	18	27
	Stadt_7.txt	20	20	23	21	21	21	22	26	23	37
	Stadt_8.txt	22	21	21	21	24	23	22	26	27	45
	<i>Summe:</i>	<i>162</i>	<i>167</i>	<i>165</i>	<b>161</b>	<i>166</i>	<i>171</i>	<i>173</i>	<i>184</i>	<i>200</i>	<i>345</i>
Fussgaenger	Fussgaenger_1.txt	4	4	2	2	3	2	2	2	4	6
	Fussgaenger_2.txt	7	8	6	8	10	8	7	8	9	11
	Fussgaenger_3.txt	2	2	3	3	4	4	5	4	6	6
	Fussgaenger_4.txt	6	4	6	5	4	5	6	8	13	14
	<i>Summe:</i>	<i>19</i>	<i>18</i>	<b>17</b>	<i>18</i>	<i>21</i>	<i>19</i>	<i>20</i>	<i>22</i>	<i>32</i>	<i>37</i>

## $d_{\max}=200$ , 8 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	13	12	11	11	11	12	12	10	28	124
	Autobahn_2.txt	12	11	11	11	11	9	9	11	24	156
	Autobahn_3.txt	7	7	7	8	5	4	9	10	8	57
	Autobahn_4.txt	12	9	11	11	11	11	11	12	25	94
	Autobahn_5.txt	10	10	10	11	10	10	9	10	21	90
	Autobahn_6.txt	12	12	15	10	10	12	14	13	30	112
	Autobahn_7.txt	11	8	8	8	8	9	7	11	22	89
	<i>Summe:</i>		<b>77</b>	<b>69</b>	<b>73</b>	<b>70</b>	<b>66</b>	<b>67</b>	<b>71</b>	<b>77</b>	<b>158</b>
Ueberland	Ueberland_1.txt	64	61	51	55	59	60	54	56	92	220
	Ueberland_2.txt	15	14	17	17	17	17	17	18	18	53
	Ueberland_3.txt	9	9	9	7	7	8	10	8	16	44
	Ueberland_4.txt	14	13	12	13	13	12	11	11	18	56
	Ueberland_5.txt	20	19	20	16	21	24	21	28	30	75
	<i>Summe:</i>	<b>122</b>	<b>116</b>	<b>109</b>	<b>108</b>	<b>117</b>	<b>121</b>	<b>113</b>	<b>121</b>	<b>174</b>	<b>448</b>
Stadt	Stadt_1.txt	9	9	9	9	8	9	9	11	16	30
	Stadt_3.txt	22	22	22	21	21	21	22	20	29	59
	Stadt_4.txt	40	40	41	41	43	41	39	43	58	79
	Stadt_5.txt	39	33	38	38	32	37	40	42	42	68
	Stadt_6.txt	14	14	14	15	12	15	17	14	21	27
	Stadt_7.txt	30	28	29	29	29	19	24	23	25	37
	Stadt_8.txt	22	22	22	22	22	22	24	24	28	45
	<i>Summe:</i>	<b>176</b>	<b>168</b>	<b>175</b>	<b>175</b>	<b>167</b>	<b>164</b>	<b>175</b>	<b>177</b>	<b>219</b>	<b>345</b>
Fussgaenger	Fussgaenger_1.txt	2	2	2	2	2	2	2	3	4	6
	Fussgaenger_2.txt	6	6	6	7	7	8	10	8	9	11
	Fussgaenger_3.txt	2	3	3	2	5	4	5	4	8	6
	Fussgaenger_4.txt	3	4	6	6	5	5	4	6	13	14
	<i>Summe:</i>	<b>13</b>	<b>15</b>	<b>17</b>	<b>17</b>	<b>19</b>	<b>19</b>	<b>21</b>	<b>21</b>	<b>34</b>	<b>37</b>

## $d_{\max}=250$ , 8 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Fussgaenger	Fussgaenger_1.txt	2	2	2	2	2	2	2	2	4	4
	Fussgaenger_2.txt	6	6	6	5	7	8	7	6	6	9
	Fussgaenger_3.txt	2	2	2	2	4	2	4	4	5	4
	Fussgaenger_4.txt	5	4	3	4	4	5	4	5	11	12
	<i>Summe:</i>	<b>15</b>	<b>14</b>	<b>13</b>	<b>13</b>	<b>17</b>	<b>17</b>	<b>17</b>	<b>17</b>	<b>26</b>	<b>29</b>

## $d_{\max}=400$ , 4 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	7	6	6	6	6	6	6	6	19	64
	Autobahn_2.txt	5	5	4	6	6	6	7	5	17	82
	Autobahn_3.txt	6	6	6	3	3	3	3	6	7	30
	Autobahn_4.txt	5	5	4	4	4	4	4	5	13	50
	Autobahn_5.txt	6	6	6	6	6	6	6	6	13	51
	Autobahn_6.txt	9	9	9	8	8	9	9	10	18	62
	Autobahn_7.txt	6	5	3	5	5	5	3	4	10	48
	<i>Summe:</i>		<b>44</b>	<b>42</b>	<b>38</b>	<b>38</b>	<b>38</b>	<b>39</b>	<b>38</b>	<b>42</b>	<b>97</b>
Ueberland	Ueberland_1.txt	36	33	31	33	34	34	39	38	46	110
	Ueberland_2.txt	12	13	13	13	11	12	12	14	15	28
	Ueberland_3.txt	3	6	4	4	4	3	4	3	9	23
	Ueberland_4.txt	8	7	6	6	6	6	8	8	13	28
	Ueberland_5.txt	9	9	9	9	11	10	11	9	17	38
	<i>Summe:</i>		<b>68</b>	<b>68</b>	<b>63</b>	<b>65</b>	<b>66</b>	<b>65</b>	<b>74</b>	<b>72</b>	<b>100</b>
Stadt	Stadt_1.txt	8	9	9	9	9	9	8	8	9	15
	Stadt_3.txt	14	14	14	16	15	13	16	11	19	31
	Stadt_4.txt	26	24	21	22	23	22	27	27	31	34
	Stadt_5.txt	22	23	23	23	24	25	26	21	28	33
	Stadt_6.txt	13	9	9	9	10	10	11	12	14	14
	Stadt_7.txt	13	10	9	10	9	11	11	11	15	19
	Stadt_8.txt	13	14	13	13	13	13	12	11	14	23
	<i>Summe:</i>		<b>109</b>	<b>103</b>	<b>98</b>	<b>102</b>	<b>103</b>	<b>103</b>	<b>111</b>	<b>101</b>	<b>130</b>
Fussgaenger	Fussgaenger_1.txt	2	2	2	2	2	2	2	2	4	2
	Fussgaenger_2.txt	5	6	5	5	5	4	5	5	5	4
	Fussgaenger_3.txt	2	2	2	2	4	1	1	1	4	3
	Fussgaenger_4.txt	3	3	3	3	3	3	3	3	7	6
	<i>Summe:</i>		<b>12</b>	<b>13</b>	<b>12</b>	<b>12</b>	<b>14</b>	<b>10</b>	<b>11</b>	<b>11</b>	<b>20</b>

## $d_{\max}=400$ , 8 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	6	6	6	6	6	6	7	8	19	64
	Autobahn_2.txt	4	4	4	4	4	5	5	5	16	82
	Autobahn_3.txt	4	4	4	4	4	4	4	5	6	30
	Autobahn_4.txt	8	8	8	9	7	9	8	9	13	50
	Autobahn_5.txt	6	6	6	6	6	6	6	6	14	51
	Autobahn_6.txt	7	6	6	6	7	8	7	8	19	62
	Autobahn_7.txt	8	7	7	8	8	8	7	7	13	48
	<i>Summe:</i>	<b>43</b>	<b>41</b>	<b>41</b>	43	42	46	44	48	100	387
Ueberland	Ueberland_1.txt	35	35	32	31	34	36	34	38	49	110
	Ueberland_2.txt	11	11	11	11	12	12	11	11	15	28
	Ueberland_3.txt	4	4	3	3	4	4	5	4	9	23
	Ueberland_4.txt	8	10	10	8	8	12	9	10	14	28
	Ueberland_5.txt	9	8	10	10	10	10	9	9	18	38
	<i>Summe:</i>	67	68	66	<b>63</b>	68	74	68	72	105	227
Stadt	Stadt_1.txt	5	8	8	8	8	8	8	8	9	15
	Stadt_3.txt	13	13	13	14	15	13	12	13	20	31
	Stadt_4.txt	24	24	25	23	27	27	24	26	31	34
	Stadt_5.txt	20	24	23	21	18	20	23	24	28	33
	Stadt_6.txt	13	8	8	11	5	5	11	8	9	14
	Stadt_7.txt	12	11	12	12	12	12	13	11	16	19
	Stadt_8.txt	14	15	13	13	13	15	15	14	15	23
	<i>Summe:</i>	101	103	102	102	<b>98</b>	100	106	104	128	169
Fussgaenger	Fussgaenger_1.txt	2	2	2	2	2	2	2	2	3	2
	Fussgaenger_2.txt	5	5	5	5	5	5	5	5	6	4
	Fussgaenger_3.txt	2	2	2	2	3	1	1	1	3	3
	Fussgaenger_4.txt	3	3	3	3	3	3	3	3	6	6
	<i>Summe:</i>	12	12	12	12	13	<b>11</b>	<b>11</b>	<b>11</b>	18	15

## $d_{\max}=500$ , 4 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	6	6	6	6	6	6	6	6	14	52
	Autobahn_2.txt	6	5	5	5	5	7	7	7	12	66
	Autobahn_3.txt	2	2	2	3	3	5	3	4	7	24
	Autobahn_4.txt	5	5	5	5	5	6	6	5	14	40
	Autobahn_5.txt	6	6	6	6	6	6	6	6	14	40
	Autobahn_6.txt	8	7	7	8	8	7	8	7	16	50
	Autobahn_7.txt	8	8	7	8	5	3	3	5	12	40
	<i>Summe:</i>		41	39	<b>38</b>	41	<b>38</b>	40	39	40	89
Ueberland	Ueberland_1.txt	30	28	30	27	22	31	28	32	38	89
	Ueberland_2.txt	8	8	8	8	8	8	8	8	11	22
	Ueberland_3.txt	3	6	3	3	3	3	3	4	6	19
	Ueberland_4.txt	6	6	5	5	5	5	5	6	10	22
	Ueberland_5.txt	6	6	10	9	10	9	9	8	13	31
	<i>Summe:</i>		53	54	56	52	<b>48</b>	56	53	58	78
Stadt	Stadt_1.txt	9	8	8	9	7	7	8	8	10	12
	Stadt_3.txt	11	11	11	11	11	11	12	12	15	23
	Stadt_4.txt	24	24	25	23	22	23	22	26	25	28
	Stadt_5.txt	15	15	14	20	18	21	19	17	24	26
	Stadt_6.txt	6	8	8	8	8	8	10	6	11	11
	Stadt_7.txt	8	8	8	8	10	9	9	10	10	14
	Stadt_8.txt	10	11	11	11	12	12	12	11	12	17
	<i>Summe:</i>		<b>83</b>	85	85	90	88	91	92	90	107
Fussgaenger	Fussgaenger_1.txt	2	2	2	2	2	2	2	2	3	2
	Fussgaenger_2.txt	4	4	4	4	4	4	4	4	4	3
	Fussgaenger_3.txt	1	1	1	1	1	1	1	1	2	2
	Fussgaenger_4.txt	3	3	3	3	3	3	3	3	3	2
	<i>Summe:</i>		<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	12	9

## $d_{\max}=500$ , 8 Regressionspunkte

Group	Filename	m_max								lin.DR	no DR
		10	20	30	40	50	60	80	100		
Autobahn	Autobahn_1.txt	6	6	6	6	6	6	6	6	16	52
	Autobahn_2.txt	6	6	6	6	7	7	7	7	12	66
	Autobahn_3.txt	5	4	4	5	5	4	3	5	7	24
	Autobahn_4.txt	7	5	6	6	5	5	8	4	16	40
	Autobahn_5.txt	6	6	6	6	6	6	6	5	13	40
	Autobahn_6.txt	5	5	5	7	6	7	8	8	17	50
	Autobahn_7.txt	2	2	2	2	2	2	2	2	10	40
	<i>Summe:</i>		<b>37</b>	<b>34</b>	<b>35</b>	<b>38</b>	<b>37</b>	<b>37</b>	<b>40</b>	<b>37</b>	<b>91</b>
Ueberland	Ueberland_1.txt	25	29	28	26	29	26	30	26	41	89
	Ueberland_2.txt	10	10	10	10	10	10	10	10	12	22
	Ueberland_3.txt	3	3	4	4	3	6	4	3	8	19
	Ueberland_4.txt	6	6	5	7	6	6	6	7	12	22
	Ueberland_5.txt	10	10	9	9	7	7	9	10	13	31
	<i>Summe:</i>		<b>54</b>	<b>58</b>	<b>56</b>	<b>56</b>	<b>55</b>	<b>55</b>	<b>59</b>	<b>56</b>	<b>86</b>
Stadt	Stadt_1.txt	7	7	7	7	8	8	7	7	9	12
	Stadt_3.txt	11	12	12	12	11	11	12	12	16	23
	Stadt_4.txt	22	22	23	22	24	25	23	24	26	28
	Stadt_5.txt	17	17	15	17	17	15	20	18	25	26
	Stadt_6.txt	6	8	8	6	6	6	11	10	9	11
	Stadt_7.txt	10	10	10	10	8	9	8	8	9	14
	Stadt_8.txt	11	11	11	10	10	10	10	11	13	17
	<i>Summe:</i>		<b>84</b>	<b>87</b>	<b>86</b>	<b>84</b>	<b>84</b>	<b>84</b>	<b>91</b>	<b>90</b>	<b>107</b>
Fussgaenger	Fussgaenger_1.txt	2	2	2	2	2	2	2	2	3	2
	Fussgaenger_2.txt	5	4	4	4	4	4	4	4	4	3
	Fussgaenger_3.txt	1	1	1	1	1	1	1	1	2	2
	Fussgaenger_4.txt	3	3	3	3	3	3	3	3	3	2
	<i>Summe:</i>		<b>11</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>12</b>

## Gesamtresultate

Nachfolgend sind die relativen Ergebnisse der Verfahren dargestellt. Dabei ist das Verfahren ohne Dead-Reckoning als Bezug verwendet. In der vorletzten Spalte steht der  $m_{max}$ -Wert, bei dem das Kartenbasierte Verfahren sein Optimum für den jeweiligen  $d_{max}$ -Wert erreicht hat. In der letzten Spalte ist die Zahl der Stützpunkte wiedergegeben, die zur Berechnung der Bewegungsrichtung verwendet wurden.

Gruppe	d_max	Kartenbasiert	Lineare Fortsetzung	kein Dead Reckoning	mit m_max	#Punkte
Autobahn	20	15,80%	22,33%	100,00%	20	4
	50	8,91%	16,71%	100,00%	20	4
	100	9,05%	18,10%	100,00%	50	4
	200	8,73%	21,47%	100,00%	50	4
	400	9,82%	25,06%	100,00%	50	4
	500	12,18%	28,53%	100,00%	50	4

Gruppe	d_max	Kartenbasiert	Lineare Fortsetzung	kein Dead Reckoning	mit m_max	#Punkte
Überland	20	24,33%	29,01%	100,00%	10	4
	50	19,29%	28,53%	100,00%	30	4
	100	19,98%	32,27%	100,00%	40	4
	200	23,44%	37,95%	100,00%	40	4
	400	27,75%	44,05%	100,00%	30	4
	500	26,23%	42,62%	100,00%	50	4

Gruppe	d_max	Kartenbasiert	Lineare Fortsetzung	kein Dead Reckoning	mit m_max	#Punkte
Stadt	20	35,52%	37,13%	100,00%	10	4
	50	37,75%	39,45%	100,00%	20	4
	100	37,63%	47,88%	100,00%	40	4
	200	46,67%	57,97%	100,00%	40	4
	400	57,99%	76,92%	100,00%	30	4
	500	63,36%	81,68%	100,00%	10	4

Gruppe	d_max	Kartenbasiert	Lineare Fortsetzung	kein Dead Reckoning	mit m_max	#Punkte
Fussgänger	20	34,33%	33,18%	100,00%	20	8
	50	39,08%	41,38%	100,00%	30	8
	100	35,29%	49,41%	100,00%	30	8
	150	38,89%	70,37%	100,00%	60	8
	200	35,14%	91,89%	100,00%	10	8
	250	44,83%	89,66%	100,00%	30	8

In der nachfolgenden Tabelle sind die relativen Ergebnisse im Vergleich zum Verfahren der linearen Fortsetzung dargestellt.

Gruppe	d_max	Kartenbasiert	Lineare Fortsetzung	kein Dead Reckoning	mit m_max	#Punkte
Autobahn	20	70,73%	100,00%	447,80%	20	2
	50	53,30%	100,00%	598,53%	20	2
	100	50,00%	100,00%	552,52%	50	2
	200	40,65%	100,00%	465,81%	50	2
	400	39,18%	100,00%	398,97%	50	2
	500	42,70%	100,00%	350,56%	50	2

Gruppe	d_max	Kartenbasiert	Lineare Fortsetzung	kein Dead Reckoning	mit m_max	#Punkte
Überland	20	83,86%	100,00%	344,73%	10	4
	50	67,61%	100,00%	350,47%	30	4
	100	61,90%	100,00%	309,89%	40	4
	200	61,76%	100,00%	263,53%	40	4
	400	63,00%	100,00%	227,00%	30	4
	500	61,54%	100,00%	234,62%	50	4

Gruppe	d_max	Kartenbasiert	Lineare Fortsetzung	kein Dead Reckoning	mit m_max	#Punkte
Stadt	20	95,68%	100,00%	269,35%	10	4
	50	95,70%	100,00%	253,52%	20	4
	100	78,59%	100,00%	208,87%	40	4
	200	80,50%	100,00%	172,50%	40	4
	400	75,38%	100,00%	130,00%	30	4
	500	77,57%	100,00%	122,43%	10	4

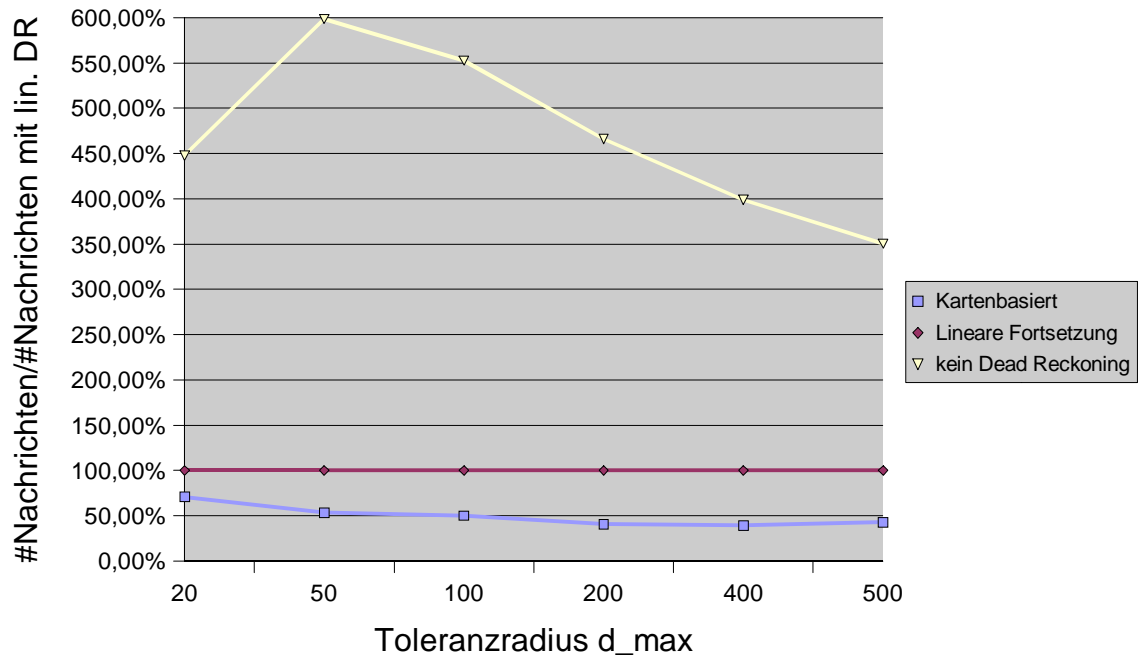
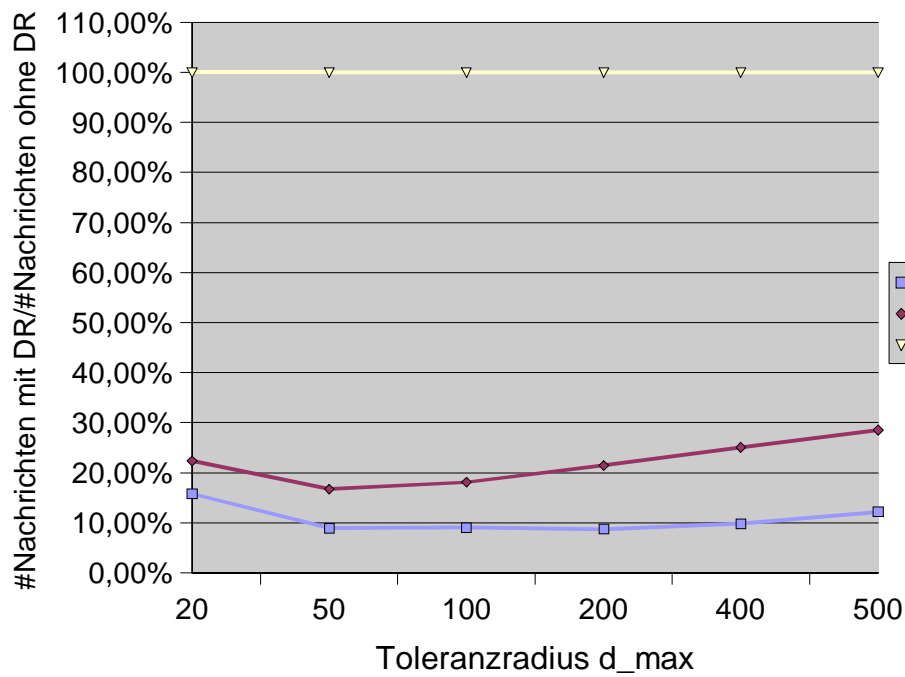
Gruppe	d_max	Kartenbasiert	Lineare Fortsetzung	kein Dead Reckoning	mit m_max	#Punkte
Fussgänger	20	103,47%	100,00%	301,39%	20	8
	50	94,44%	100,00%	241,67%	30	8
	100	71,43%	100,00%	202,38%	30	8
	150	55,26%	100,00%	142,11%	60	8
	200	38,24%	100,00%	108,82%	10	8
	250	50,00%	100,00%	111,54%	30	8

In der letzten Tabelle ist die Updaterate dargestellt. Der jeweilige Wert zeigt wie viele Updatenachricht das jeweilige Verfahren pro Stunde verschickt.

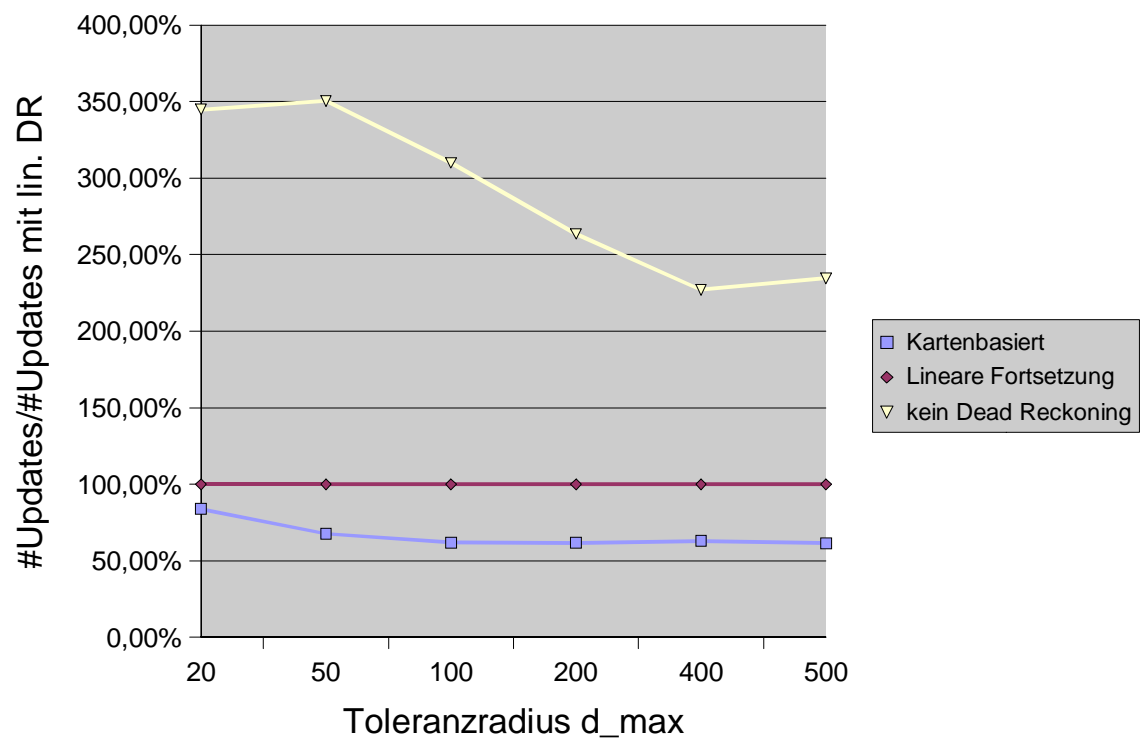
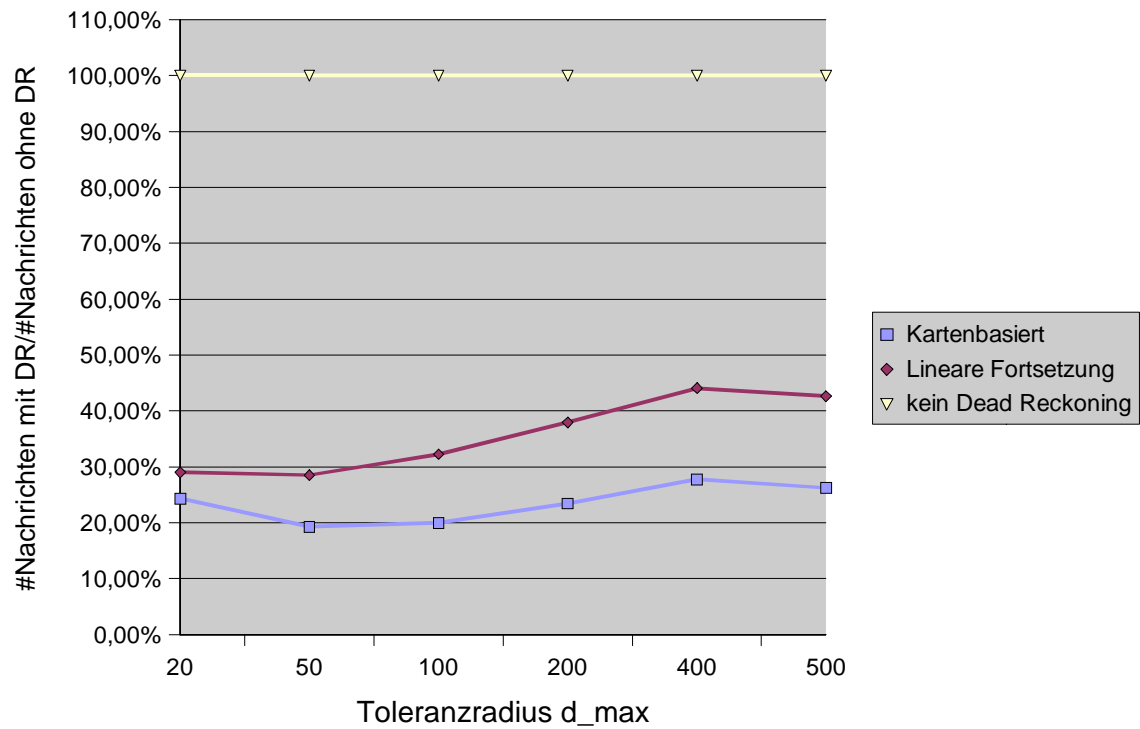
<b>Updates/h</b>				
<b>Gruppe</b>	<b>d_max</b>	<b>Kartenbasiert</b>	<b>Lineare Fortsetzung</b>	<b>kein Dead Reckoning</b>
Autobahn	20	367,48	519,54	2326,5
	50	138,12	259,13	1551
	100	75,4	150,79	833,16
	200	39,92	98,2	457,44
	400	24,08	61,46	245,2
	500	24,08	56,39	197,68
<b>Gruppe d_max Kartenbasiert Lineare Fortsetzung kein Dead Reckoning</b>				
Überland	20	453,79	541,15	1865,52
	50	174,72	258,44	905,76
	100	102,53	165,62	513,25
	200	63,7	103,13	271,79
	400	38,22	60,67	137,71
	500	29,12	47,32	111,02
<b>Gruppe d_max Kartenbasiert Lineare Fortsetzung kein Dead Reckoning</b>				
Stadt	20	404,06	422,31	1137,5
	50	203,27	212,4	538,47
	100	106,61	135,65	283,34
	200	66,79	82,97	143,12
	400	40,65	53,93	70,11
	500	34,43	44,39	54,34
<b>Gruppe d_max Kartenbasiert Lineare Fortsetzung kein Dead Reckoning</b>				
Fussgänger	20	69,71	67,37	203,04
	50	31,81	33,68	81,4
	100	14,04	19,65	39,77
	150	9,82	17,78	25,26
	200	6,08	15,91	17,31
	250	6,08	12,16	13,57

# Diagramme

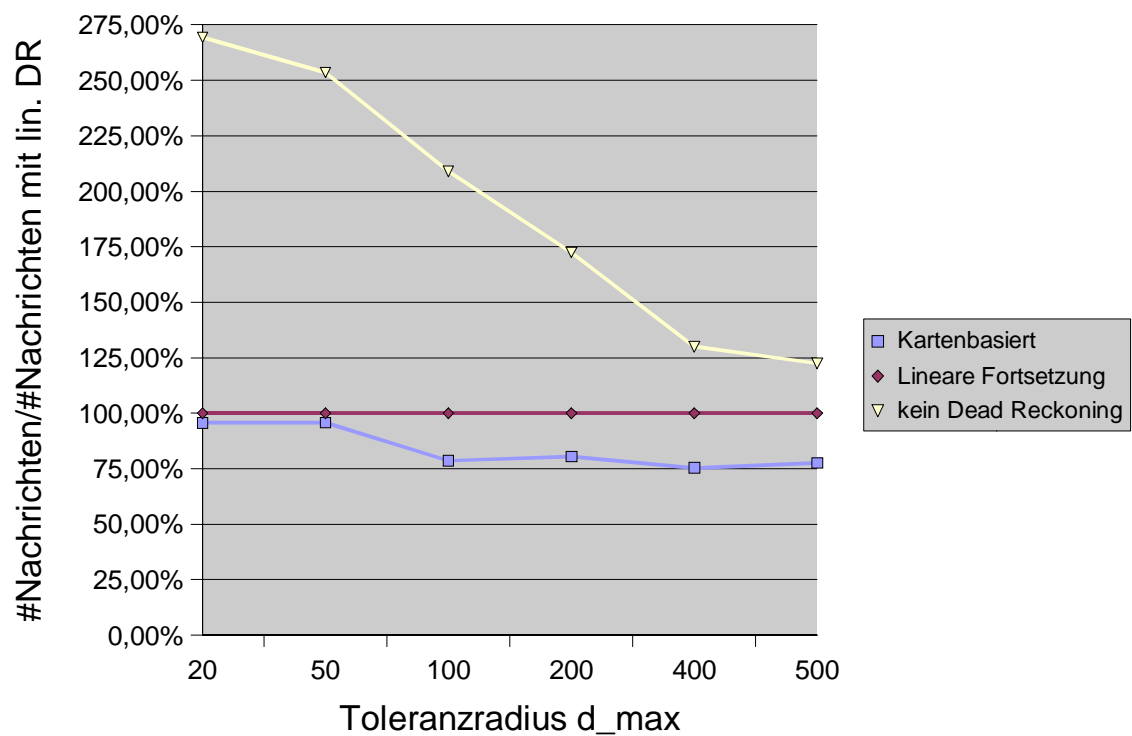
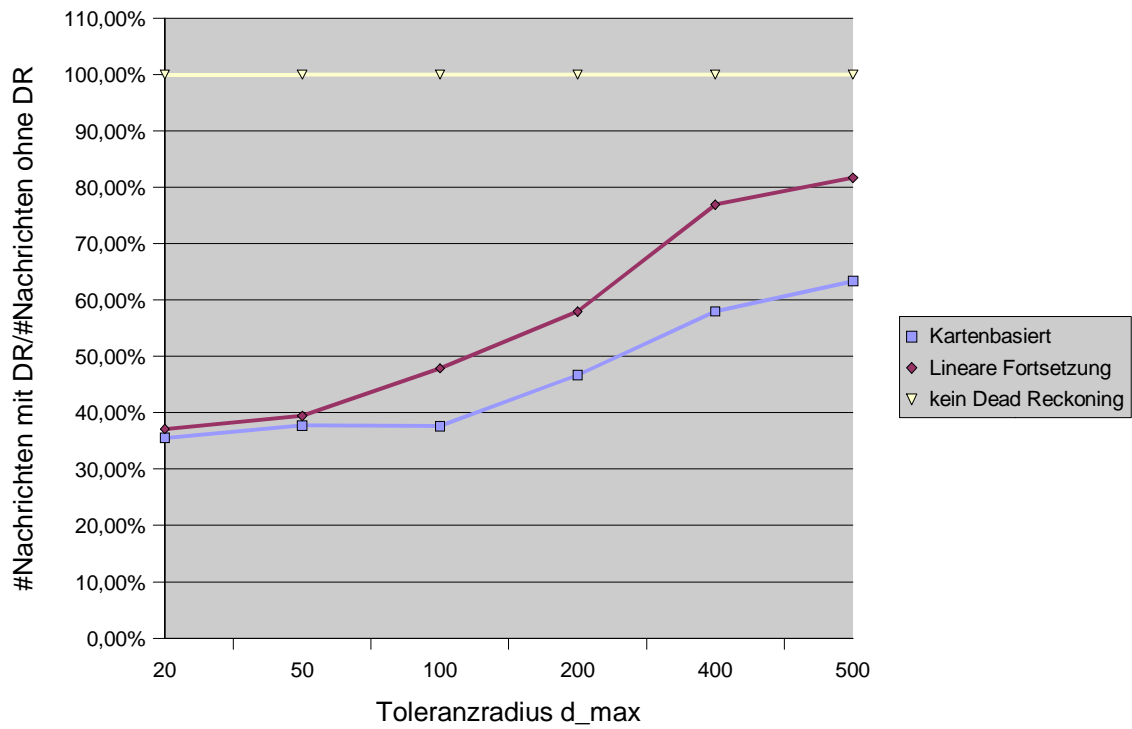
## Autobahnfahrt



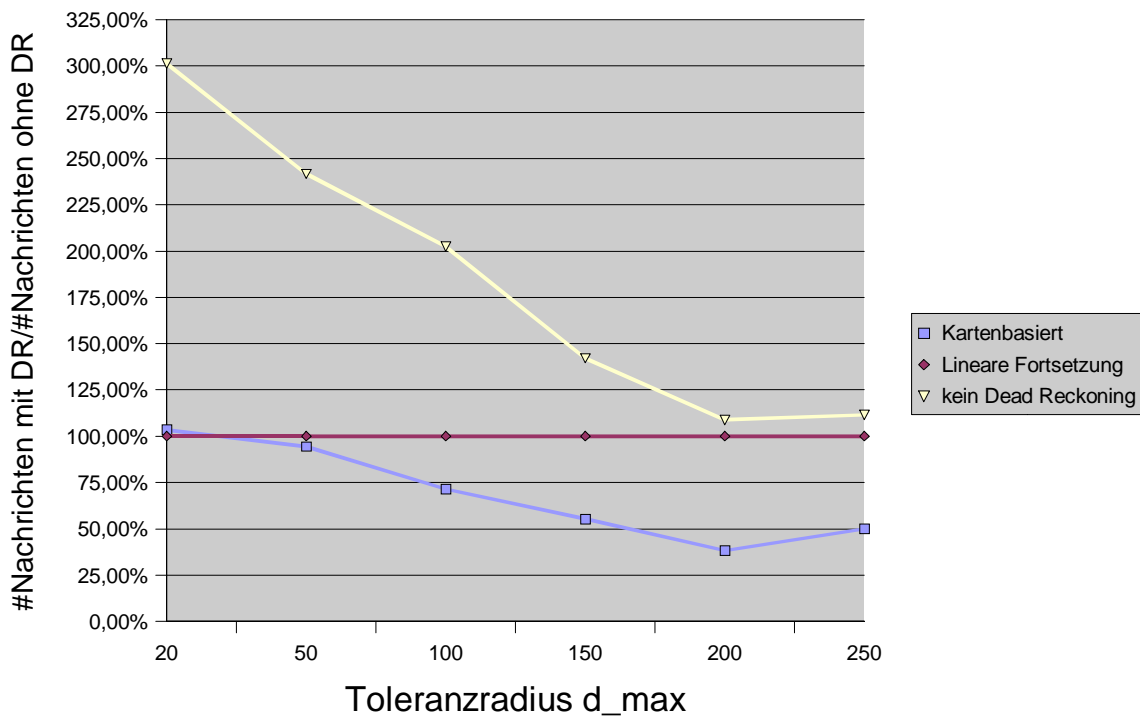
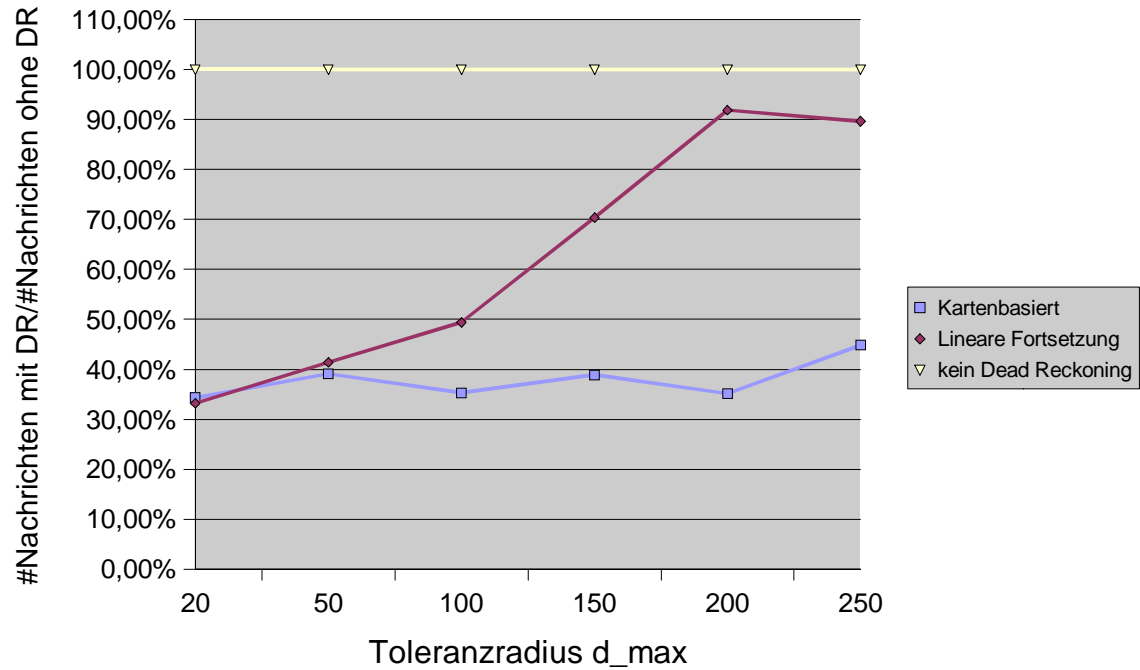
## Überlandfahrten



## Stadtfahrten



## Fußgänger



Achtung: Die Diagramme für Fußgänger haben andere Werte für  $d_{max}$  als die anderen Schaubilder! Der Grund ist, daß einzelnen die Traces nicht lang genug waren, um sinnvolle Werte zu simulieren, die größer sind! Daher wurden nur kleinere Werte betrachtet.

Ich versichere, daß ich diese Arbeit selbständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.