GIS für Fahrzeugnavigation

Dr. Ing. Claus Brenner, SS05

Zusammenfassung

Jan Hinzmann

12. Oktober 2005

Inhaltsverzeichnis

1 Einleitung

In der heutigen Fahrzeugnavigation geht es darum, einen Nutzer mit Hilfe von digitalen Karten in seiner Wegfindung zu unterstützen.
In dieser Zusammenfassung werden die drei Themenbereiche *Digitale Karte*, *Routensuche* und *Ortung und Kartenstützung* besprochen.

2 Digitale Karte

2.1 Motivation

Digitale Karten werden in den verschiedensten Bereichen verwendet, z.B.

- Routenplanung
- Zielführung
- Kartenunterstützung
- Anzeige auf Displays
- Adressauflösung (Ort, Straße, Hausnummer → Positsion in der Karte)
- Informationen über POIs (Point of Interest)

Geographische Daten werden in einer Datenbank gehalten, im Anwendungsfall für die Navigation sind besonders die Straßen von Interesse. Eine Straßen-DB besteht dann aus

- Geometrie
- Topologie
- sonstige Informationen

Um nun die digitalen Karten zu benutzen sind Informationen über ihre Beschaffenheit notwendig, wozu insbesondere die Darstellung der *Geometrie* und *Topologie* gehören, welche im Folgenden näher besprochen werden sollen.

2.2 Geometrie

Die Geometrie der geographischen Daten kann durch elliptische oder ebene Koordinaten dargestellt werden.

2.2.1 Ellipsoidische Koordination

Es gibt verschiedene Arten von elliptischen Koordinaten

- lokale, bestanliegende
- konventionelle
- mittleres Ellipsoid

  der Mittelpunkt liegt in der Mitte der Erde und die Achse entspricht der Rotationsachse der Erde

Heute ist für die Fahrzeugnavigation aufgrund der GPS-Nutzung das WGS 84 (World Geodetic System 1984) üblich. WGS ist ein globales Referenzsystem.

2.2.2 Ebene Koordinatensysteme

Als ebenes Koordinatensystem ist in Deutschland das Gauss-Krüger System üblich. Hierbei wird mit einer transversalen Mercator-Projektion gearbeitet und einem 3° Meridianstreifen, um den Fehler am Rand zu minimieren.
Ansonsten ist der *universal transversal Mercator* (UTM) verbreitet, welcher ähnlich zu Gauss-Krüger ist, aber mit 6° breiten Streifen arbeitet.
Die ebenen Koordinatensysteme werden immer dann benötigt, wenn die Karte oder GPS (WGS 84) mit der Koppelnavigation (Länge, Winkel) kombiniert werden soll. Allerdings ergeben sich hierbei folgende Probleme:

- Die Positionsgleichungen sind aufwendig
- Da an den Streifenrändern Probleme auftreten, wurde eine Überlappung eingeführt.

Da häufig nur ein kleiner Teil der Karte für das Fahrad interessiert, wird oftmals mit primitiven Kantenprojektionen gearbeitet.

2.2.3 Repräsentation der Geometrie

Die Geometrie kann entweder durch *Rasterkarten* oder durch *Vektorkarten* repräsentiert werden. Rasterkarten sind aber eher unüblich.
Vektorkarten enthalten Knoten und Kanten. Knoten werden durch Koordinaten repräsentiert (λ, Φ). Kanten spannen sich zwischen Knoten auf und evtl. werden noch Hilfsknoten (shape points) ergänzt. Aus den Knoten und Kanten resultiert schließlich eine Polygonrepräsentation.

2.3 Topologie

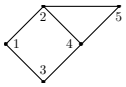
Die Topologie beschäftigt sich mit der Lage der Kanten bzw. Knoten zueinander. Durch Graphen kann die Topologie repräsentiert werden. Hierbei wird zwischen

- ungerichteten Graphen,
- gerichteten Graphen und
- gewichtete (bewertete) Graphen

unterschieden.

Im Allgemeinen ist ein Graph ein Tupel $G = (V, E)$, wobei V die Menge der Knoten und E die Menge der Kanten ist. V und E sind disjunkt. Für E gilt: $E \subseteq [V]^2$, wobei $[V]^2$ die Menge der zweielementigen Teilmengen von V ist.

Beispiel Sei $G_1 = (V, E)$ ein Graph mit $V = \{1, 2, 3, 4, 5\}$
und $E = \{\{1, 3\}, \{1, 2\}, \{3, 4\}, \{2, 4\}, \{2, 5\}, \{4, \}\}$.



Dieser Graph hat unterschiedliche Repräsentationen, da es keine absoluten Positionsangaben für die Knoten gibt. Eine Möglichkeit der Repräsentation ist durch das obige Bild gegeben.

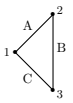
Man spricht davon, dass ein Graph *planar* oder eben ist, wenn sich seine Kanten nicht schneiden. Da das Straßennez nicht planar ist, lässt es sich nicht durch planare Graphen repräsentieren.

Gerichtete Graphen werden definiert durch: $G = (V, E, \Phi, \Psi)$, wobei $\Phi : E \to V$ die Menge der Anfangsknoten und $\Psi : E \to V$ die Menge der Endknoten ist.

Beispiel Sei $G_2 = (V, E)$ ein zweiter Graph, mit $V_2 = \{1, 2, 3\}$ und $E_2 = \{A, B, C\}$. Ferner sind folgende Relationen gegeben:

- $\Phi(A) = 1$, $\Psi(A) = 2$
- $\Phi(B) = 2$, $\Psi(B) = 3$
- $\Phi(C) = 1$, $\Psi(C) = 3$

So ergibts sich beispielsweise der folgende Graph:



Hier erhalten die Kanten Namen, damit sie anschließend referenziert werden können. Werden den Kanten weitere Attribute, wie etwa (Länge, Typ, Zeitbedarf, ...) zugeordnet, so spricht man von *gewichtete* oder auch bewerteten Graphen.

Um nun noch Nachbarschaftsverhältnisse ausdrücken zu können, kennt die Topologie die Begriffe *Adjazenz* und *Inzidenz*, wobei Adjazenz die Nachbarschaft gleicher Elemente und Inzidenz die Nachbarschaft verschiedener Elemente beschreibt.
Der *Grad eines Knotens* ist charakterisiert durch die Zahl seiner inzidenten Kanten.

Eine Adjazenzmatrix ist symmetrisch und kann beispielsweise durch ein zweidimensionales Array mit booleschen Werten repräsentiert werden. Diese Struktur dient zur Speicherung. Eine Listenstruktur lohnt sich erst ab 64 Einträgen pro Zeile.

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

	n_1	n_2	n_3	n_4	n_5
e_1	1	1	0	0	0
e_2	0	0	1	1	0
e_3	1	0	1	0	0
e_4	0	1	0	1	0
e_5	0	1	0	0	1
e_6	0	0	0	1	1

Bei einer Inzidenzenmatrix sind den Zeilen bzw. Spalten die Namen der Kanten bzw. Knoten zugeordnet. Ausserdem muss eine Inzidenzenmatrix nicht symmetrisch sein.

2.4 Beispiel: GDF

2.4.1 Historie

- GDF = Geographic Data File
- GDF 1.0 = Draft 1988 als Ergebnis con EUREKA/REMETER. Basierend auf *UK National Transfer Format*.
- GDF 2.0 = entwickelt von *European Digital Road Map Task Force* EDRM: Daimler Benz, Bosch, Philips, Renault, ...
- GDF 2.1 (Okt. 92), 2.2 (Nov. 94), 3.0 (Okt. 95), ...
- inzwischen GDF 5.0

GDF dient als Standard für die Darstellung und Übertragung von Daten über

- Straßen
- Verwaltungsgebieten
- Siedlungen
- Landnutzung
- Brücken, Tunnel
- Eisenbahnen
- Gewässer
- Verkehrszeichen
- Services (Tankstellen, ...)
- öffentliche Transportmittel

2.4.2 Features, Attributes, Relationships

Die Feartures, Attributes und Relationships stellen zusammen die Infrastuktur dar.

Feature Stellt ein Objekt der realen Welt dar (Straße, Kreuzung, Verwaltungsgebiet, ...).

Attribute Stellt Eigenschaften von Objekten oder Relationen dar (Namen, Anzahl Spuren, Straßenlampen, ...).

Relationship Stellen Beziehungen zwischen Objekten her (Straße innerhalb eines Gebietes, Gegenstände auf der Fahrbahn, Abbiegeverbote, ...).

2.4.3 Exkurs: NIAM-Diagramme

NIAM = Nijssen's Information Analysis Methodology

//FIXME HIER MÜSSEN DIAGRAMMTYPEN EINFGEFÜGT WERDEN

2.4.4 Datenmodell von GDF

Siehe hierzu Ausdruck „Mini-Zusammenfassung über GDF“

2.4.5 Features

Features sind in GDF zu sogenannten *„Feature Themes“* gruppiert und innerhalb dieser Gruppen gibt es *„Feature Classes“*.

Bsp.:

Theme	„Straßen und Fähren“ (Themecode: 41)
Class	4110 Road Element
	4120 Junction
	4130 Ferry Connection
Theme	„Eisenbahn“ (Themecode: 42)
Class	4210 Railway Element
	4220 Railway Elementjunction

Weiter gibt es *„Feature Categories“*, die die Typen `Point, Line, Area, Complex` definieren. Durch das *„Feature Representation Theme“* wird eine Zuordnung

`Feature Classes` → `Feature Categories`

getroffen.
So ist zum Beispiel bei dem Classen

4110	Road Element
4120	Junction

die Klasse `4110` ein *linefeature* und die Klasse `4120` ein *pointfeature* usw.

In GDF werden 3 Level unterschieden: Level0, Level1, Level2.

Level0 – Geometrie

Dieser Level repräsentiert die Geometrie. Hierbei ist ein *Node* ein Punkt mit `x,y,z`-Koordinaten und eine *Edge* eine Kante mit Verweisen auf Anfangs- und Endknoten und `x,y,z`-Koordinaten der Shapepoints. Schließlich ist ein *Face* ein Fläche mit Verweisen auf umgebene Kanten.
Level0 ist ein panarer Graph.

Level1 – Simple Features

In diesem Level geht es um *Simple Features*, wie Punkte, Linien, Flächen, die eine Bedeutung erhalten.

Level 1		**Level 0**
Point	korrespondiert mit	Node
Line	korrespondiert mit einer / mehrerer	Edge(s)
Area	korrespondiert mit einer / mehrerer	Face(s)

Level 1 ist nicht planar (=glättbar).

Level2 – Complex Features

Complex Features bestehen aus *Simple Features* (Level1) oder sind selbst *Complex Features*.

Beispiele:

Intersection (Level 2) besteht aus Road Elements (Level 1) und Junctions Level 1)

Road (Level 2) besteht aus Road Elements (Level 1)

Level 2 ist nicht planar.

2.4.6 Attributes

Attribute bilden die Eigenschaften von realen Objekten ab. Beispiele:

Attribut	**Beispiel**	**Code**
Official Name	"GER//Manstraße"	ON
Alternate Name	"GER//Braunschweigerplatz"	AN
Brand Name	SShell"	BN
Blocked Passage	0 = nein 1 = blockiert am Anfang 2 = blockiert am Ende	BP
Direction of Traffic Flow	1 = offen in beide 2 = geschlossen pos. 3 = geschlossen neg. 4 = geschlossen in beide	DF
First House No left		LS
First House No right	„UND 9a“	RS
Last House No left	undetermined	LE
Last House No right		RE
Route Number	„UND A2“	RN
Telephone Number	„UND +(49)-(511)...“	TL
Vehicle Type	0 = alle 11 = PKW 12 = Anwohner	VT
Functional Road Class	0 - 9	FC

Attribute können „segmentiert“ angenommen werden, wenn sie nur für einen Teilbereich gültig sind.

Punkt-, Linien- und Flächendeatures können eine Liste von Attributen beinhalten und so gibt es Verweise von den Features zu den Attributen.

2.5 Relationships

Die Relationships modellieren die Beziehungen zwischen den Objekten. Dabei kann ein Feature Teil von mehreren Beziehungen sein und zwei Features

können in verschiedenen Relationships involviert sein. Dabei kann eine Beziehung binärer Natur sein oder mehrere Features einbeziehen. Wie auch schon die Attributes, so haben auch die Relationships einen Namen und einen Code:

Code	**Name**	**Beteiligte Features**
1001	Roadelemnt in Administrative Area	Order & Area Roadelement
1011	Roadelement in Built-up Area	Built-up Area Roadelement
1022	Service along Roadelement	Roadelement Service
2103	Prohibited Manoevre	From Roadelements via Junction via Roadelement ⋮ To Roadelement

Verweise von Relationships zu Features...

2.5.1 Die physikalische Darstellung von GDF

GDF in ASCII-Dateien (ISO 8859-1, Latin 1) abgelegt. Ein Land z.B. Deutschland besteht aus Sammlungen (Alben) von Dateien (Volumen). In den Dateien befinden sich sog. *„Records“* z.B. Line Feature ist in einem Record abgelegt (= logical record). Die Speicherung von Records erfolgt zeilenweise, je 80 Zeichen für einem sog. *„media record“*.
Eine solche Recordzeile wird mit `0` terminiert, wenn die Zeile endet und mit `1`, wenndie Zeile fortgesetzt werden muss. Am Anfang jeder Zeile steht ein Typecode für den folgenden Record, oder ein Fortsetzungszeichen (continuation mark = 00).

0	1	2	3	4	...	78	79
5	2				1 logischer Record = 1 media Record		0
⋮					⋮		

Tabelle 1: einzeiliger GDF Media Record

Die Zahl `52` am Anfang der Zeile steht für den Typecode des Records. In Spalte 79 steht eine 0, was zeigt, dass es sich um einen einzeiligen Record handelt.

Beispiel 2: Der logische Record in Tabelle 2 besteht aus 3 Mediarecords und terminiert so nicht mit einer 0 am Ende der ersten Zeile. Er signalisiert durch die 1 in der letzten Spalte, dass er fortgeführt wird. So beginnen alle nachfolgenden zugehörigen Zeilen mit der *Continuationmark* 00, bis der logische Record in der dritten Zeile in Spalte 79 mit einer 0 endet.

Die Daten innerhalb eines Records sind durch Felder und Längen genau festgelegt. So hat zum Beispiel ein Noderecord die Struktur aus Tabelle 3.

0	1	2	3	4	...	78	79
5 0 0	2 0 0				1 logischer record mit 3 media records		1 1 0
⋮					⋮		

Tabelle 2: mehrzeiliger GDF Media Record

0 1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 20 1 2 3 4	...	78 79
2 5 0 0 2 0 0 0 0 0 1	0 0 2 0 0 0 0 0 0 0 5 0 ...	... 2 ...	0

Tabelle 3: mehrzeiliger GDF Media Record

Die Records werden in Definitionen beschrieben, wo die Länge und Werte der einzelnen Felder in einem Record beschrieben sind.
Beispiel:

Filedname	**Size**	**Type**	**NoData**	**Beschreibung**
REC-DESC	2	N	Obl	Rec-Type Code = 25
KNOT-ID	10	N	<s>	Node-ID
YXZ-ID	10	N	<s>	Geometry ID
FACE-ID	10	N	<s>	Face ID
STATUS	2	N	<s>	Status Code; 1=Section border, 2=Normal

Tabelle 4: Definition des Node Records (25)

Wobei in der Spalte *Size* die Länge eines Feldes angegeben wird. Durch * kann eine freie Länge gewählt werden.
Für die Spalte *Type* sind folgende Datentypen vorgesehen:

- N = Numerisch
- I = Vorzeichen numerisch
- A = Alphabetisch
- AN = Alphanumerisch
- G = Bed. Zeichen

In der Spalte *NoData* wird die Kardinalität angegeben, wobei `Obl` obligatorish meint.

Die Reihenfolge von Records in GDF ist festgelegt:

1. Header: Volume, Daten, Section, Layer Header
2. Records für Koordinaten, Nodes, Edges, Faces, Points, Lines, Areas.d

Die Verweise zwischen den einzelnen Records erfolgen über IDs..

2.6 Weitere Aspekte digitaler Straßenkarten

2.6.1 Erzeugung von Straßen-DB

Aus Analogen Karten, Luft- und Orthobildern, sowie Feldbegehungen werden von Kartenanbietern wie etwa TeleAtlas, NavTech Kartendatenbanken erstellt. Die Navigationsgeräteherrsteller konvertieren diese Daten, die in GDF geliefertwerden, unter zuhilfenahme von Informationen verschiedener Reiseführer in verschiedene Endprodukte, wie navigierbare Karten, Darstellbare Karten, Hierarchische Adressstrukturen oder andere Executables, die auf CDs bzw. DVDs gespeichert werden.

Je nach Land, Gebiet und Kartenhersteller liegt ein verschiedener Erfassungsgrad vor:

- Teile nicht erfasst
- Teile mit niedriger Genauigkeit erfasst
- Fehlende Attribute

2.6.2 Kachelung & Clustering von Karten

Das Problem ist, dass die Gesamtdaten auf Grund ihrer Größe und der benötigten Zeit nicht in den Hauptspiecher geladen werden können. Also liegt eine Unterteilung in kleinere Einheiten nahe.
Hierbei kommen verschiedene Strategien und Datenstrukturen zum Einsatz:

Quadtree Diese Struktur ist gekennzeichnet von einer nicht überlappenden, rekursiven Unterteilung des Datenraums. Die Unterteilungen sind dabei pro Ebene gleich groß.

k-d Bäume Diese Struktur ist gekennzeichnet von einer nicht überlappenden, rekursiven Unterteilung. Die Unterteilungen sind dabei nicht gleich groß. Im zugehörigen Baum werden abwechselnd die x und y- Koordinaten gespeichert.

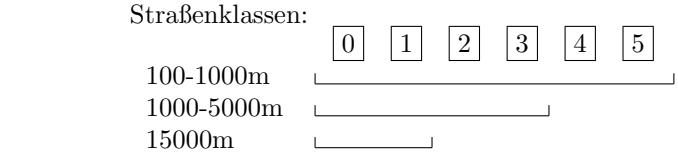
R-Bäume Diese Struktur ist gekennzeichnet durch sich überlappende Rechtecke, weöche ihre Entsprechung in einer Baumstruktur finden.

Straßencluster Diese Struktur ist gekennzeichnet von einer nicht überlappenden Unterteilung, deren Grenzen vom Straßennetz gebildet werden. Hier findet man Straßen, die scheinbar ein Gebiet zusammenstellen.

2.6.3 Hierarchische Karten

Die Auswahl der geeigneten Datenstruktur hängt im wesentlichen von dem Nutzungsprofil ab. Soll zum Beispiel ein „Zoom out“ auf ganz Deutschland realisiert werden, wäre ein Quadtree ungeeignet (6-7 Mio Kanten).
Als **einfaches Vorgehen** für die Visualisierung eignet sich die Einteilung der Straßen in Klassen. Dann werden verschiedene Layer erzeugt, die Untermengen der Straßen enthalten. Bei Auswahl eines Maßstabes, wird der entsprechende Maßstab gelesen.

Straßenklassen:



Der Nutzer stellt nun ein Anfrage an das System in einem bestimmten Maßstab und aus diesem werden die entsprechenden Daten herausgegeben.
Ein Nachteil ist, dass es so keine Generalisierung gibt. Der Schwarzwald würde verschwinden.
Ein **komplexeres Vorgehen** berücksichtigt die Generalisierung. Hier stellt sich das Problem, dass die Routensuche stark abhängig von der Anzahl der zu besuchenden Kanten ist. Ausserdem werden die Kanten nicht unbedingt weniger, beim Weglassen von Straßenklassen (vgl. Abb. 1).

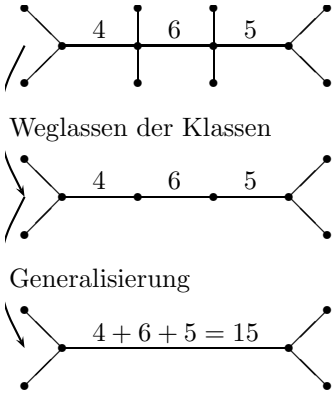


Abbildung 1: Verschiedene Generalisierungsschritte/-möglichkeiten

Die Generalisierung kann sich zuweilen als schwierig erweisen. Man stelle sich beispielsweise ein Autobahnkreuz vor.

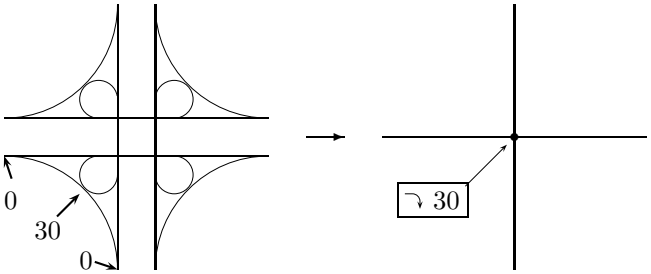


Abbildung 2: Generalisierung von GDF Level1 (links) nach GDF Level2 (rechts)

2.6.4 Offene Probleme

Offene Probleme sind beispielsweise die *Verschmelzung von verschiedenen Karten*. Sollen Karten über Grenzen hinweg dargestellt werden, so kommen die zugrundeliegenden Karten aus versciedenen Ländern. Hier können dann Inkonsistenzen am Kartenrand auftreten.
Bei der sog. *geometrischen Diskontionuität*, die zwischen Karten auftreten kann, wird versucht, das Zusammenführen zu automatisieren (*„Seaming“*).

Objekte, die nicht in der Karte auftreten, müssen georeferenziert werden. Beispielsweise kommt es bei Parkhäusern zu Problemen, da das System den kürzesten Weg routet, hier aber evtl. kein Eingang vorhanden ist. In der Datenbank eines anderen Datenanbieters ist der Weg hingegen verzeichnet.
Eine Lösung für dieses Problem wäre eine Beschreibung der Anfahrt unabhängig vom Kartenanbieter. Hierzu können Hotels, Parkhäuser, ... „Geostummel“ (z.B. auf ihren Homepages) anbieten. Dieser Stummel wird dann von einem Kunden heruntergeladen und in das Navigationssystem eingebunden.

Qualitätsaussagen sind ebenfalls problematisch, z.B. die Genauigkeit der Erfassung.

Bisher gibt es keine Höhenangaben.

Zusätzliche Daten könnten z.B. für 3D-Stadmodelle dienen.

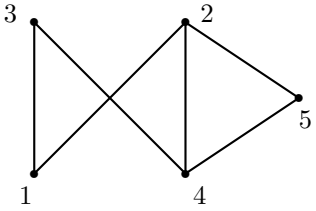
3 Routensuche

Das Kernproblem stellt sich folgendermaßen dar:
Ausgehend von einem Startknoten soll das Navigationssystem den kürzesten (besten) Weg zu einem Zielknoten finden.

3.1 Algorithmen für unbewertete Graphen

3.1.1 Adjazenzmatrix - Multiplikation

Der gezeigte Graph soll im Folgenden als Beispiel dienen.



Für ihn wird eine Adjazenzmatrix konstruiert, in dem man die Knoten auf beiden Achsen einer Tabelle aufträgt und an die Stellen eine 1 einträgt, wo eine direkte Verbindung besteht. Die Adjazenzmatrix enthält dann Informationen, welche anderen Knoten man von einem gegebenen Knoten mit einem Schritt erreichen kann.

↓

A	1	2	3	4	5
1		1	1		
2	1			1	1
3	1			1	
4		1	1		1
5		1		1	

Über eine gewöhnliche Matrizenmultiplikation $A \cdot A$ erhält man die Matrix A^2, welche dann Informationen enthält, welche Knoten man von einem Startknoten i mit 2 Schritten erreichen kann.

Die normale Matrixmultiplikation $C = A \cdot B$ ist durch $c_{ij} = \sum_k a_{ik} \cdot b_{kj}$ gegeben, und die Adjazenzmatrixmultiplikation wird durch $C = A \times B \qquad c_{ij} = \bigvee_k a_{ik} \wedge b_{kj}$ definiert.

Führt man diese Multiplikation nun auf die konstruierte Matrix A aus, so ergibt sich Matrix A^2 folgendermaßen:

A^2	1	2	3	4	5
1	1	0	0	1	1
2	0	1	1	1	1
3	0	1	1	0	1
4	1	1	0	1	1
5	1	1	1	1	1

Hierbei wurden leere Felder mit 0 aufgefüllt. Ein Element A_{ij} ergibt sich nach obiger Vorschrift durch die „Veroderung“ der i-ten Zeile mit der j-ten Spalte der Ausgangsmatrix A. Dies geschiet im Einzelnen durch die „Verundung“ der Elemente dieser Zeile bzw. Spalte.

Diese Adjazenzmatrix (A^2) hilft nun Aussagen zu treffen, welche Knoten man mit 2 Schritten erreichen kann.
Es gilt also bis jetzt:

- $A_{ij} = 1 \Rightarrow$ man gelangt von Knoten i nach Knoten j in einem Schritt.
- $A^2_{ij} = 1 \Rightarrow$ man gelangt von Knoten i nach Knoten j in zwei Schritten.

Allgemein gilt:

- $A^n_{ij} = 1 \Rightarrow$ man gelangt von Knoten i nach Knoten j in n Schritten.

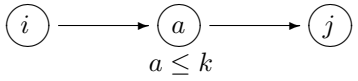
Will man nun die Frage beantworten, auf welchem Pfad man von Knoten i nach Knoten j in *höchstens* n Schritten kommt, so verodert man die einzelnen Adjazenzmatrizen: $A \vee A^2 \vee A^3 \vee ... \vee A^n$. Im gegebenen Beispiel ergibt sich, dass

man jeden Knoten in höchstens zwei Schritten erreichen kann.

3.1.2 Algorithmus von Warshall

Die Berechnung von Entfernungen mit Hilfe von Adjazenzmatrizen hat eine Komplexität von $O(n^4)$. Dies ist sehr schlecht und eignet sich deshalb nur für kleine Graphen. Durch eine andere Betrachtungsweise gelangt man zu einem viel besseren Algorithmus, nämlich dem von Warshall.

Die Idee ist hierbei, im k-ten Schritt die Matrix $A[i, j; k]$ zu berechnen, sodass gilt: $A[i, j; k] = 1 \Leftrightarrow$ Es gibt einen Pfad von i nach j auf dem höchstens Knoten der Nummer k besucht werden.



Algorithmus von Warshall (berechnet die transitive Hülle des Graphen):

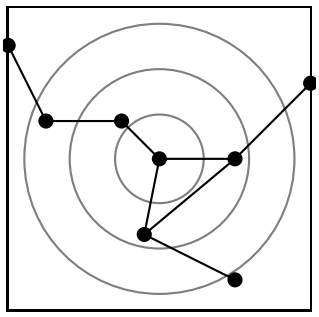
- Initialisierung: Adjazenzmatrix $A[i, k; 0] = A$
- Iteration: $A[i, j; k] = A[i, j, k-1] \vee$
- Algorithmus:

```
A[i,k;0]=A //(Adjazenzmatrix)
for k=1 to n do
    for i=1 to n do
        for j=1 to n do
            A[i,j] = A[i,j] or (A[i,k] and A[k,j];
        done
    done
done
```

Dieser Algorithmus berechnet ebenfalls alle Wege zwischen den Knoten, aber mit einer Komplexität von $O(n^3)$.
Es gibt eine graphische Vorgehensweise, in der klar wird, dass im Schritt k die Zeile k und die Spalte k identisch sind. Dann gilt das Absorbtionsgesetz $(a \vee (b \wedge a) = a)$ und es gilt $A[k, i; k] = A[k, j; k-1]$.
//FIXME Beispiel ...

3.1.3 Breitensuche

Die bekannteste Anwendung der Breitensuche ist der Dijkstra Algorithmus.

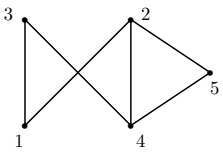


Bei der Breitensuche werden zunächst alle kurzen Wege untersucht und der Algorithmus findet so schnell einen passenden. Bei der Tiefensuche, werden alle Wege untersucht, d.h. auch alle langen Wege. Diese sind bei der Navigation aber nicht von Interesse. Man berechnet also zunächst die Erreichbarkeit von Knoten ausgehend von einem Startknoten. Anschließend traversiert man den Graphen ausgehend vom Startknoten in konzentrischen Kreisen, bis der Zielknoten erreicht ist.
Als Datenstruktur bieten siche hier Schlangen (Queue, FIFO) an. Diese haben die Methoden `rein(k)` und `n = raus()`.

Der Algorithmus der Breitensuche ist dann der folgende:

```
Startknoten s;
markiere(s);
rein(s);

while(Schlange nicht leer)
do
    k = raus();
    for (alle nicht markierten Nachbarn n von k)
    do
        markiere n;
        rein(n);
    done
done
```



Für den Beispielgraph un den Startknoten 1 liefert der Algorithmus in der Initialisierung: | 1 | | und in der Iteration folgt dann | 1 | 2 | 3 | |
//FIXME Bilder und Stacks
Durch hinzufügen eines Zählers lässt sich die Distanz zu allen Knoten ermitteln (in Zahl der Kanten).
Es ergibt sich eine Komplexität von $O(k)$, die Zahl der Kanten.

3.2 Algorithmen für bewertete Graphen

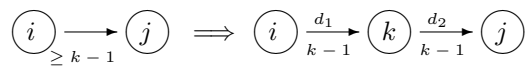
3.2.1 Algorithmus von Floyd

Die Idee von Floyd ist ähnlich zu der von Warshall, aber hier werden Distanzen berücksichtigt:

Statt: $A[i, j;\, k] = A[i, j;\, k-1] \vee (A[i, k;\, k-1] \wedge A[k, j;\, k-1])$

gilt Jetzt: $A[i, j;\, k] = min\,(A[i, j;\, k-1], A[i, j;\, k-1] + A[k, j;\, k-1])$

Graphisch: In der Initialisierung werden für $A[i.j;\, o]$ die Distanz $d[i,j]$ gesetzt,

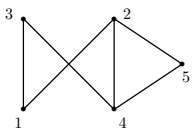


wenn es einen Weg dmit er Distanz d gibt. Gibt es keinen Weg, dann wird der Wert auf ∞ gesetzt.
Der Allgorithmus berechnet dann die Distanz von jedem Knoten zu jedem anderen Knoten und die Komplexität ist $O(n^3)$.

Beispiel (Die leeren Zellen enthalten ∞):

$A[i.j;\, 0]$

	1	2	3	4	5
1		1	1		
2	1			1	1
3	1			1	
4		1	1		1
5		1		1	



Der Algorithmus merkt erst später, dass es keinen kürzeren Weg gibt.

$A[i.j;\, 1]$

	1	2	3	4	5
1		1	1		
2	1	2	2	1	1
3	1	2	2	1	
4		1	1		1
5		1		1	

$A[i.j;\, 2]$

	1	2	3	4	5
1	2	1	1	2	2
2	1	2	2	1	1
3	1	2	2	1	3
4	2	1	1	2	1
5	2	1	3	1	2

$A[i.j;\, 3] = A[i,j;\, 2]$

$A[i.j;\, 4]$

	1	2	3	4	5
1	2	1	1	2	2
2	1	2	2	1	1
3	1	2	2	1	2
4	2	1	1	2	1
5	2	1	2	1	2

$A[i.j;\, 5] = A[i,j;\, 4]$
Der Algorithmus berichtigt die Distanzen (von 3 auf 2).

3.2.2 Algorithmus von Dijkstra

Dijkstra hatte eine Idee, ähnlich der Breitensuche, allerdings werden hier Knoten in aufsteigender Distanz zu einer Menge hinzugefügt.

Dabei wird die Menge aller Knoten in zwei Teilmengen `Optimal` und `Rest` zerlegt. Die Menge `Optimal` beinhaltet alle Knoten, zu denen die optimale Distanz bereits gefunden ist und `Rest` ist die Menge der Knoten, zu denen noch keine Distanz gefunden worden ist.

In jedem Schritt wird ein Knoten k aus `Rest` mit optimaler Distanz gewählt. Dann wird dieser Knoten aus `Rest` entfernt und zu `Optimal` hinzugefügt. Anschließend werden die Distanzen zum Nachbarn korrigiert.

Allgemein:

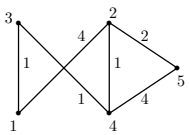
```
Startknoten s;
Knotenmenge k;
Optimal opti = {s};
Rest rest = k/{s};

//init
for ( k aus rest ) do
    D[k] = d[s, k] wenn es einen Weg gibt
         = \infty , sonst
done;

//iterate
while ( rest nicht leer ) do
    wähle k aus rest mit min(D[k]);
    opti += {k}
    rest  = rest - {k}
    for ( alle Knoten n von k ) do
        D[n] = min(D[n], D[k] + d[k,n])
    done;
done;

done;
```

Beispiel für den Dijkstraalgorithmus.



Für ein Beispiel des Dijkstra-Algorithmus dient wieder der Beispielgraph. Allerdings sind diesesmal die Kanten gewichtet (Kante $\{1, 2\}$ hat z.B. die Gewichtung 4). Es sei der Startknoten $S = 1$, dann ergeben sich für den Anfang die Mengen `Optimal` O und `Rest` R zu $O = \{1\}$, $R = \{2, 3, 4, 5\}$. Der Algorithmus startet und füllt eine Tabelle zeilenweise mit den Distanzen (entspr. den Gewichtungen) vom Startknoten zu allen anderen Knoten in `Rest`. Die Distanz für nicht

erreichbare Knoten wird mit ∞ angenommen, so ergibt sich die erste Zeile der Tabelle.

Opt	D[2]	D[3]	D[4]	D[5]	Rest
$\{1\}$	4	$\boxed{1}$	∞	∞	$\{2,3,4,5\}$
$\{1,3\}$	4	–	$\boxed{2}$	∞	$\{2,4,5\}$
$\{1,3,4\}$	$\boxed{3}$	–	–	6	$\{2,5\}$
$\{1,3,4,2\}$	–	–	–	$\boxed{5}$	$\{5\}$
$\{1,3,4,2,5\}$	–	–	–	–	$\{\emptyset\}$

Anschließend wird der Knoten, der die kleinste Distanz aufweist bestimmt, zu O hinzugefügt und aus R entfernt. Nach der ersten Runde ergibt sich, da Knoten 3 die kleinste Distanz zum Startknoten 1 hat, $O = \{1,3\}$, $R = \{2.4.5\}$. Nun fährt der Algorithmus fort und bestimmt wieder die Distanzen vom der Menge `Optimal` zu allen Knoten in `Rest`. Während der Algorithmus läuft, ergeben sich zunächst auch Distanzannahmen, die später korrigiert werden, da dann Knoten zu `Optimal` hinzukommen, die kürzere Wege erlauben. Diese werden dann auch gefunden.

Der Dijkstra-Algorithmus funktioniert und findet die kürzesten Distanzen zu allen anderen Knoten, wenn keine negativen Distanzen zugelassen sind (wichtig). Für die Navigation kann man ihn abbrechen, wenn der Zielknoten gefunden, also zu `Optimal` hinzugefügt worden ist. Es gibt dann keinen besseren (kürzeren) Weg.

Angenommen, im Schritt j wird Knoten k zu `Optimal` mit einer Distanz $D[k]$ hinzugefügt. Gäbe es einen kürzeren Weg $(s) \rightsquigarrow (k)$ mit $D[k] < \tilde{D}[k]$, so kann dies kein direkter Weg sein, weil er dann schon erkannt worden wäre.
Und ein Knoten i, über den es einen kürzeren Weg gibt, kann nun in zwei Fällen existieren:

Fall 1: Der Knoten i ist schon in `Optimal`
Kann nicht sein, da er in Schritt j mit $\tilde{D}[k]$ hinzugefügt worden wäre.

Fall 2: Der Knoten i ist in `Rest`
Dann muss der Weg über i immer ein Umweg sein:

$$\begin{aligned} D[i] &\geq D[k] \\ \tilde{D}[k] &= D[i] + d[i,k] \\ &\geq D[k] + d[i,k] \\ &\geq D[k] \end{aligned}$$

Der entstehende Suchraum gestaltet sich üblicherweise in Form eines Kreises/ Ellipse, mit dem Radius $d[s,z]$, wobei s =Startknoten und z =Zielknoten.
Wenn alle Kantengewichte identisch sind, dann entspricht der Dijksta-Algorithmus der Breitensuche.

Für die Suche nach dem optimalen Weg führt man nun einen Rückwärtszeiger ein – so wird der Weg navigierbar.
Hierbei wird der Code

```
...
    for ( alle Knoten n von k ) do
        D[n] = min(D[n], D[k] + d[k,n])
    done;
...
```

in der Iteration durch

```
...
    for ( alle Knoten n von k ) do
        if (D[k] + d[k, n] < D[n]) do
            D[n] = D[k] + d[k,n];
            Vorgänger[n] = k;
        done;
    done;
...
```

ersetzt. Die Tabelle ergibtsich dann zu:

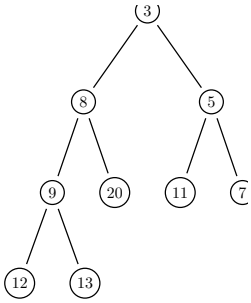
Opt	D[2]	D[3]	D[4]	D[5]	Rest
$\{1\}$	4_1	$\boxed{1}_1$	∞	∞	$\{2,3,4,5\}$
$\{1,3\}$	4_1	–	$\boxed{2}_3$	∞	$\{2,4,5\}$
$\{1,3,4\}$	$\boxed{3}_4$	–	–	6_4	$\{2,5\}$
$\{1,3,4,2\}$	–	–	–	$\boxed{5}_2$	$\{5\}$
$\{1,3,4,2,5\}$	–	–	–	–	$\{\emptyset\}$

Der optimale Pfad von $\textcircled{1} \rightsquigarrow \textcircled{5}$ mit $D[k] < \tilde{D}[k]$ ist dann also

$$5_2 \to 2_4 \to 4_3 \to 3_1 \to 1$$
$$1, 3, 4, 2, 5$$

Die Zeitkomplexität des Dijkstra-Algorithmus mit e Kanten und n Knoten ergibt sich in einer einfachen Implementierung zu

- Initialisierung $O(n)$
- Iteration $O(n^2)$, n mal wird ein Knoten k gewählt und für jedes k wird das Minimum bestimmt.



Beispielgraph eines HEAPs

Als Datenstruktur für die Speicherung bietet sich ein $HEAP$ an, einem Binärbaum mit dem kleinsten Element an der Wurzel. In einem HEAP kann man mit $O(n \log n)$ suchen und er wird durch ein eindimensionales Array a abgebildet. In $a[0]$ steht die Wurzel des Baumes, in $a[1]$ der linke und in $a[2]$ der rechte Kindknoten. Alle weiteren linken Kindknoten stehen dann in $a[2i]$ und alle rechten in $a[2i+1]$, wobei i die Stufe im Baum angibt. Neue Elemente werden an der Wurzel eingebunden und *„sinken dann ein“*. Dies ist in $O(\log n)$ möglich. Das entsprechende Array für die Speicherung des obigen HEAPs sieht dann folgendermaßen aus:

$$\boxed{3}_0\boxed{8}_1\boxed{5}_2\boxed{9}_3\boxed{20}_4\boxed{11}_5\boxed{7}_6\boxed{12}_7\boxed{13}_8$$

Insgesamt ergibt die Zeitkomplexitätsbetrachtung für den Dijkstra-Algorithmus mit der Datenstruktur $HEAP$ für die Speicherung und ebenen Graphen zu $O(n \log n)$, was nur schwach überlinear ist. (Super!) Deshalb ist Dijkstra optimal, da das Problem nicht besser als $O(n \log n)$ lösbar ist.

3.3 Suchraumexplosion und informative Suchverfahren

Beim Dijkstra-Algorithmus wird zwar der beste Weg gefunden, es werden aber auch alle Knoten expandiert. Dies kann unter Umständen zuviel sein (Speicher, Zeit, ...). Deshalb wurden Algorithmen entwickelt. die dieses Problem berücksichtigen.

3.3.1 (kombinatorische) Explosion des Suchraums

Gehen beispielsweise von jedem Knoten b Kanten ab, so stellt sich die Anzahl der untersuchten Knoten durch

$$s = 1 + b + b^2 + b^3 + ... + b^k = \sum_{i=0}^{k} b^i$$

dar. Es handelt sich also um ein exponentielles Wachstum. (Das ist tödlich)

3.3.2 „Best First“ und die Form des Suchraums

Es soll der bester Knoten zuerst expandiert werden. Daruch soll sich die Form des Suchraums verbessern und weniger unnötige Knoten expandiert werden.
Der Dijkstra-Algoritmus wählt den Knoten mit der kürzesten Distanz zum Startknoten, diese Funktion heiße $g(n)$. So wird aber nur berücksichtigt, wie weit es von *Start* bis n ist und nicht, wie weit es noch bis zum Ziel ist. Deshalb entsteht die erweiterte Form des Suchraums. Es sind also alle Knoten in einem Umkreis mit dem Radius $d[s, z]$ expandiert.

//FIXME graph + tabelle (es werden auch unsinnige Knoten expandiert)

3.3.3 Informative Suche: Greedy-Verfahren

Greedy = „gierig“
Für die Routensuche lässt sich die Luftlinie durch eine Funktion $h(n)$ berechnen, welche dann als Indikator dient, um in jedem Schritt den Knoten zu wählen, der die Luftliniendistanz am meisten verringert.

//FIXME graph + tabelle (es werden keine unsinnigen Knoten expandiert)
Allerdings ist die berechnete Route nicht immer optimal.

Allgemein Die Greedy-Verfahren finden mit geringem Aufwand „gute“ Lösungen, jedoch nicht unbedingt optimale.

3.3.4 Informative Suche: Das A*-Verfahren

Die beiden vorangegangenen Abschnitte haben gezeigt:

- $g(n)$ viele Schritte, optimal
- $h(n)$ wenig Schritte, nicht optimal

Beim A^*-Verfahren werden nun beide Funktionen benutzt, um die richtigen Knoten zu wählen. Die Funktion ist dann $F(n) = g(n) + h(n)$, wobei $h(n)$ noch eine Restschätzung darstellt, damit die tatsächliche Entfernung der Knoten zum Ziel immer $\geq h(n)$ ist. Dies ist möglich, da die Luftliniendistanz eine zusätzliche Restlängenschätzung darstellt.
//FIXME graph + tabelle

Allgemein Das A^*-Verfahren findet generell die optimale Lösung.
//FIXME noch mehr theorie ...
Je besser die Restschätzung h, desto kleiner der Suchraum, da die Anzahl der expandierten Knoten von der Differenz der Restschätzung und der tatsächlichen Distanz abhängt.
Es gilt hier einen Kompromiss zu finden, denn wenn die Restschätzung exakt ist (h = Luftliniendistanz), werden Umwege, die aber beispielsweise zur Autobahn führen, nicht berücksichtigt. Auf der anderen Seite ($h = 0$ entspr. Dijkstra) werden alle Knoten expandiert, was zu Explosion des Suchraums führt.

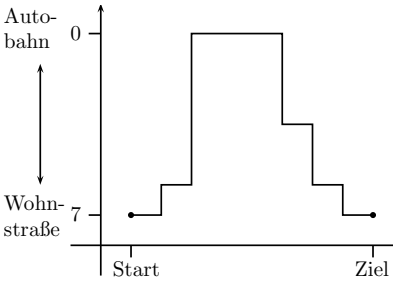
3.4 Weitere aspekte der Routensuche

3.4.1 Zweiseitige Suche

Die Idee ist hier, nicht nur vom Start zum Ziel zu suchen, sondern gleichzeitig auch umgekehrt. Wie bei den obigen Verfahren, soll diese Herangehensweise den Suchraum weiter verkleinern.
Dafür gibt es allerdings keine Garantie, wenn es Umwege gibt, treffen sich die beiden Suchräume nicht.
Auch ist eine Suche vom Ziel zum Start nicht immer möglich, man denke beispielsweise an zeitlich beschränkte Einbahnstraßen.

3.4.2 Heuristik: Straßenklassen

Die Heuristik - Lehre von der Auffindung wissenschaftlicher Erkenntnisse auf methodischem Weg.



Das Problem ist, dass selbst beim A*-Verfahren noch zuviele Knoten expandiert werden. Da eine typische Route allerdings in ihrem Verlauf in der Mitte oft nur Autobahnen benötigt, brauchen dort kleine Straßen nicht berücksichtigt zu werden. Man kann also ein Konzept von Straßenklassen benutzen, wobei den Autobahnen die kleinste Klasse (0) zugeordnet wird und Wohnpfaden höhere Klassen. Ein Algorithmus wird dadurch allerdings nicht optimal, und eventuell wird keine Lösung gefunden. Die Lösung hierfür sind denn Netzwerkklassen.

3.4.3 Hierarchische Karten

Verwendet man Kartendaten, die nur der Straßenklasse des momentan expandierten Knoten entsprechen, so müssen nicht benötigte Kartendaten nicht geladen werden. Dies stellt dann eine enorme Zeitersparnis und somit einen Performanzgewinn dar. Heut sind 4 Level üblich

3.4.4 Generalisierung

Hier spielt das *„Traveling Sailsman“*-Problem eine Rolle. Dabei gilt es, eine geschlossene Route durch mehrere Punkte (Städte) zu finden, sodass der Weg möglichst kurz wird. Dieses Problem ist NP-vollständig und es müssen insgesamt $(n-1)!$ Kanten untersucht werden, wenn n Knoten (Städte) besucht werden sollen.

Auch gibt es das sog. *vehicle routing problem*, das Problem der Flottenplanung. Hier benötigen n Kunden gewisse Mengen der Ware $A, B, C, ...$ und es gibt m Fahrzeuge, die eingesetzt werden können. Die Fahrzeuge haben gewisse Attribute, wie max. Beladung, Max. Strecke. Die Aufgabenstellung ist es dann, die Gesamtkosten zu optimieren.

3.4.5 Kantenorientiertes Routing

Da die erlaubten Manöver in der Regel von den Kanten (Straßen) abhängen, sind die Routingverfahren kantenorientiert, statt Knotenorientiert.

4 Ortung und Kartenstützung

4.1 Ortungsverfahren für KFZ

Es gibt verschiedene Ortungsverfahren:

- Sterne

- Lotsen: Erkennen von Landmarken
- Koppelnavigation: Anfangsposition + Richtung, dann Abfolge von Distanzen + Richtungen

  schlecht: Fehler addieren sich
- Inertialnavigation: Afangsposition + Richtung + Geschwindigkeit, dann Abfolge von Beschleunigungen und Richtungen

  schlecht: Fehleraddition
- Funkortung: Laufzeit- und Phasenmessung zu Sendern mit bekannten Positionen

  terrestrisch: DECCA, LORAN-C

4.2 Sensoren für die Ortung im KFZ

Sensoren für eine Ortung in Kraftfahrzeugen lassen sich in *absolut messende* und *relativ messende* unterteilen.

- absolutmessend

  GPS

  Kompass
- relativmessend

  Odometer

  Differentielles Odometer (Man schließt aus der Differenz der Drehung der Räder auf Kurven)

  Drehratensensor (Gyro)

4.2.1 GPS als absoluter Sensor

Das Global Positioning System GPS funktioniert durch Satelliten in der Umlaufbahn, die Zeitsignale (Atomuhren) senden. Ein Empfänger vergleicht dieses Signal mit seinem eigenen Zeitsignal (Quarzuhr) und errechnet daraus seine Position. Hierzu benötigt er mindestens drei Satelliten. Da die Quarzuhren der Empfänger in der Regel nicht ausreichen, um die Laufzeiten der Satellitensignale korrekt zu berechenen, wird noch ein vierter Satellit benötigt.
Man erziehlt heutzutage eine Genauigkeit von $100m$ mit SA und $10m$ ohne SA. SA bedeutet Selective Availability, eine künstliche Verschlechterung des Signals (USA, historische, militärisch-taktische Gründe). Mit dem Satellitenpositionierungsdienst des AdV (Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland) (SAPOS) lässt sich eine Genauigkeit von $1 - 3cm$ erzielen und werden die Daten noch nachbearbeitet (Postprocessing), so kommt man auf $< 1cm$.

Güteaussage:

- Zahl der Satelliten
- Konstellation

- Dilation of precision

Probleme:

- Abschattung in Städten
- Mehrwegeausbreitung durch Reflektion führt zu falschen Positionen ohne Hinweis
- berechnete Position mit Zeitverzögerung ($\approx 1s$)
- Startverhalten (Zeit, bis erste Lösung)

4.2.2 Kompass als absoluter Sensor

Der Kompass arbeitet mit Hilfe einer Magnetfeldonde und hat eine hohe Anfälligkeit gegenüber Störungen wie

- Magnetfeldern im KFZ
- elektr. Felder (Heckscheibe,Laufsprecher)
- andere Fahrzeuge, Stromleitungen, Stahlstrukturen (Brücken, ...)

4.2.3 Odometer

Richtungsänderungen werden können durch ein Odometer bestimmt werden. Dazu wird am Getriebe des Fahrzeug optisch oder induktionell die Richtungsänderungen abgenommen. Der Vorteil bei dieser Methode ist die Störunanfälligkeit und es lassen sich relativ genaue Ergebnisse erzielen ($\leq 0.5\%$, also 5m Fehler auf 1km).
Probleme bereiten der sog. Schlupf (Nässe, Eis, Beschleunigung) und eine ungenaue Winkelmessung.

4.2.4 Drehratensensor (Gyroskop, Gyro)

Optische oder mechanosche Kreisel sind zu teuer, deshalb werden Vibrations-Gyroskope oder Mikromechanik eingesetzt. Dieses System wird meistens nur für die Longitudinalachse (Hochachse) eingesetzt.
Beim Vibrationsgyroskop wird eine Stimmgabel in Schwingung versetzt und die Coriolis-Kraft erzeugt messbare Querschwingunen. Problematisch ist hier der sog. Bias (Sensordrift/Offset), welcher aber an der Ampel kalibriert werden kann. Ausserdem problematisch ist die Nichtlinearität und ein schnelles Wachstum der Fehler bei Koppelnavigation.

4.3 Kombination verschiedener Messverfahren

4.3.1 Üblich für KFZ

Im Auto werden oft alle drei Möglichkeiten eingesetzt, also

$$GPS + Odometer + Gyroskop,$$

da bei der ausschließlichen Nutzung von GPS z.B. in Tunneln die Navigation ausfallen würde. In der Kombination erhält man die Genauigkeit von Odometer + Gyro und die Langzeitstabilität von GPS. Auch lassen sich die Sensoren gegnseitig kalibrieren.

4.3.2 Einführende Beispiele

//FIXME Ausgleichsrechnung

Beispiel Inkrementelle Bestimmung eines Mittelwertes mittels *kleinster Quadrate Methode*
...

4.3.3 Kalman-Filterung

Der Kalmenfilter ist ein stochastischer Zustandsschätzer, für dynamische Systeme.
Der Filter dient dazu, Zustände oder Parameter eines dynamischen Systems durch reduntante Messungen zu schätzen. Dabei wird der mittlere quadratische Fehler minimiert.
Da er iterativ arbeitet, eignet er sich hervorragend für die Anwendung in Echtzeitsystemen. Die Berechnung der Werte findet dabei in zwei Schritten statt

- Prädiktion
- Korrektur