1 Einführung

Durch neue Technologien in ihren Möglichkeiten voran gebracht, ist die Positionsbestimmung heutzutage für viele Anwendungsgebiete interessant und findet auch immer weitere Verbreitung.
In dieser Studienarbeit wird im folgenden auf eine Form der Positionsbestimmung eingegangen, die in den letzten Jahren für jeden verfügbar wurde, nämlich die Bestimmung der Position mit Hilfe von GPS (Global Positioning Service). Dessen Funktionsprinzip ist die Positionsbestimmung mittels Satelliten, die die Erde in etwa 17500 Kilometern Höhe umkreisen. Mit Hilfe dieser Satelliten, genauer ihrer bekannten Position zu einem bestimmten Zeitpunkt, läßt sich die Position äußerst präzise bestimmen. Durch die erst kürzlich durchgeführte Abschaltung der sogenannten *Selective Availability,* einer künstlich herbeigeführten Verschlechterung der zu erlangenden Positionsschärfe, wurde die Menge der möglichen Anwendungsgebiete des GPS-Systems sogar noch erweitert.
Die Aufgabenstellung dieser Arbeit bestand nun darin, die Kommunikation eines (D)GPS-Empfängers mit einem Rechner unter Windows 95/98 zu ermöglichen. Mit anderen Worten war eine Kommunikations-Bibliothek zu erstellen, mit deren Hilfe Daten aus einem OEM-Board, einem (D)GPS-Empfänger ohne weitere Funktionalität, zu erhalten sind. Als Programmiersprache war Java 1.2 vorgegeben. Für die Kommunikation mit einem solchen Empfänger über eine serielle Schnittstelle steht das von der *National Marine Electronics Association* entwickelte *NMEA*-Protokoll zur Verfügung. Dieses Standardprotokoll basiert auf dem Versenden und Empfangen von Datenrahmen, die Zustands- und Steuerinformationen enthalten. Ein weiterer Punkt der Aufgabenstellung war, daß der Zielrechner ein sogenannter *Wearable Computer* der Firma Xybernaut sein sollte. Diese Art Rechner haben die Besonderheit, daß sie so tragbar sind, daß für eine Bedienung keine Hand notwendig ist.
Um die Funktionalität der erstellten Bibliothek zu demonstrieren, sollte außerdem eine Beispielsanwendung, die die wichtigsten über die Schnittstelle erhaltenen Navigationsdaten anzeigen kann, und ein Tool, mit dem auf einer grafischen Karte die aktuelle Position eingezeichnet werden kann, entwickelt werden.

Die vorliegende Arbeit ist in folgende Kapitel untergliedert. Die Funktionsweise des GPS-Systems, sowie dessen Besonderheiten und mögliche Verbesserungen werden in Kapitel 2 näher dargestellt. In diesem Kapitel werden außerdem einige Anwendungsgebiete beschrieben, die dem Leser die vielfältigen Möglichkeiten, für die GPS nutzbar ist, aufzeigen sollen. Um die durch eine Positionsbestimmung erhaltenen Positionsdaten richtig einordnen und anwenden zu können, ist es nötig, einige Grundbegriffe der Positionsbestimmung zu verstehen. Das im Umfang dieser Arbeit relevante Basiswissen ist in Kapitel 3 zusammengestellt. Nach den theoretischen Grundlagen der Kapitel 2 und 3 wird in Kapitel 4 der Entwurf der Kommunikationsbibliothek beschrieben werden. Hier werden neben den Vorgaben und Anforderungen an eine solche auch die auftretenden Einschränkungen, die sich z.B. aus der Verwendung des *NMEA*-Protokolls ergaben, und ihre Konsequenzen für den endgültigen Aufbau der Kommunikationsbibliothek dargestellt. Implementierungstechnische Details, eine Beschreibung der einzelnen öffentlichen Programmierschnittstellen und der wichtigsten internen Abläufe, die bei der Datenermittlung mit der Kommunikationsbibliothek auftreten, werden in Kapitel 5 gegeben. Eine Erklärung des Konzepts und der Bedienung der Beispielsanwendung und des grafischen Tools wird dann in Kapitel 6 geliefert. In Kapitel 7 werden Auszüge aus vorgenommenen Tests dargestellt, die vom Empfänger erhaltene Daten in Relation zu Umgebungsvariablen, wie z.B. der Bewölkung oder der Nähe hoher Gebäude, stellen und so beispielhaft die Bedeutung und Beeinflußbarkeit dieser Daten veranschaulichen. Kapitel 8 schließt mit einer Zusammenfassung dieser Arbeit und einem Ausblick auf mögliche Anwendungen die Ausarbeitung ab.

2 Global Positioning Service (GPS)

In diesem Kapitel wird das System des *Global Positioning Service* oder kurz *GPS* genauer erklärt. Dazu wird zuerst die Motivation für die Entwicklung von GPS dargestellt. Anschließend wird die Funktionsweise des *GPS-Systems* kurz umrissen. Im folgenden Unterkapitel wird die Funktionsweise dann detaillierter erklärt, indem die einzelnen Schritte, die von GPS bei der Bestimmung einer Position durchgeführt werden, beschrieben werden. Ein Unterkapitel über das NMEA-Protokoll erklärt dann, auf welche Weise Daten von den Empfängern zum Verbraucher gelangen können und wie diese dabei kodiert werden. Dieses Kapitel abschließen wird eine Aufzählung und kurze Erklärung einiger Einsatzgebiete für GPS. Dieses Kapitel orientiert sich in Inhalt und Aufbau an [TRI97], Beispiele sind, wenn nicht anders angegeben hieraus entnommen.

2.1 Motivation für die Entwicklung des GPS-Systems

Schon immer war das Wissen, wo man sich gerade befindet und wohin man sich bewegt, einer der großen Wünsche, aber auch eine große Notwendigkeit der Menschen. Die Kunst der Navigation und vor allem der Positionsbestimmung war und ist für eine große Menge von Tätigkeiten immens wichtig, wenn nicht sogar überlebenswichtig. Man denke hierbei z.B. an die Flugnavigation oder das Wandern durch weites, unbewohntes Land.
Allerdings war es früher schwer und umständlich, verläßliche Positionsdaten zu erhalten. Ein weiteres Problem waren dabei auch die Genauigkeitsgrenzen, die sich aus den einzelnen Verfahren zur Positionsbestimmung ergaben und die viele mögliche Anwendungsgebiete, die ein hohes Maß an Präzision erforderten, ausschlossen. Über die Jahre wurden viele Techniken entwickelt, doch jede hatte außer den Vorteilen auch wieder Nachteile gegenüber den anderen.
Diese Techniken und Nachteile waren unter anderem:

- *Navigation mit Hilfe bekannter Wegpunkte* (z.B. Leuchttürme oder markante Felsformationen).
  Nachteile: Sind nur lokal zu gebrauchen, sind außerdem Umwelteinflüssen unterworfen und können sich daher verändern oder ganz verschwinden.
- *Navigation mit der Dead Reckoning Technik*
  Nachteile: Kompliziert, die Genauigkeit hängt ganz entscheidend von den verwendeten Navigationswerkzeugen ab, die relativ ungenau sind. Außerdem akkumulieren Fehler sehr schnell, da die aktuelle Messung ja auf vorherige Messung Bezug nimmt.
- *Navigation anhand des Sternenfirmament*s (z.B. Nordstern als Fixpunkt).
  Nachteile: Kompliziert, nur möglich bei Nacht und klarem Himmel, ist also sehr zeit-/umweltbeinflußt.
- *Das OMEGA-System* (wurde mittlerweile aufgegeben).
  Positionsbestimmungen waren anhand empfangener langwelliger Radiosignale von 8 auf der Erdoberfläche verteilten Sendestationen möglich. Aus dem Empfang mehrerer dieser Funkfeuer ließ sich dann die eigene Position berechnen.
  Nachteile: Positionsbestimmung basiert auf relativ wenigen Funkfeuern, die Genauigkeit ist daher beschränkt und außerdem Funkstörungen unterworfen.
- *Das LORAN-System*.
  Das LORAN-C System wurde entwickelt, um die Navigation in Küstennähe zu verbessern. Dazu wurden entlang der Küstenlinien Sendestationen installiert, die Radiosignale ausstrahlten. Durch den Empfang mehrerer dieser Radiosignale konnte die eigene Position berechnet werden.

Nachteile: Nur geringe Überdeckung (praktisch bloß an Küsten einsetzbar), schwankende Genauigkeit in Abhängigkeit der geografischen Lage. Leicht zu stören.

Die Nachteile jeder dieser bestehenden Alternativen und das Verlangen nach einem verläßlichen und präzisen System zur Positionsbestimmung seitens des Militärs veranlaßte das US-Verteidigungsministerium (engl.: Department of Defense, kurz DoD) die Entwicklung eines neuen Systems in Auftrag zu geben, das all die oben erwähnten Nachteile nicht mehr haben würde. Daraufhin wurde das heute als Global Positioning Service (GPS) bekannte System geplant, entwickelt und installiert, ein Unterfangen das insgesamt ungefähr 12 Milliarden US-Dollar kostete. Durch die Freigabe des Systems für zivile Anwendungen wurde GPS das heute am weitesten verbreitete System zur Positionsbestimmung.

2.2 Übersicht GPS

GPS ist ein weltweit einsetzbares, funkgestütztes Navigationssystem, das seine Funktionalität aus der Konstellation von 24 Satelliten und einigen Bodenstationen bezieht. Diese Satelliten umkreisen die Erde in einer Höhe von ungefähr 20.200 Kilometern zweimal am Tag , wobei die Dauer einer einzelnen Erdumkreisung 11 Stunden und 58 Minuten beträgt. Die Satellitenkonstellation ist dabei so beschaffen, daß insgesamt 6 Orbitalbahnen mit jeweils 4 Satelliten bestehen.Die Satelliten werden vom GPS-System als Referenzpunkte verwendet, mit denen es technisch möglich ist, die Position bis auf weniger als einen Zentimeter genau zu bestimmen. Die Funktionsweise dieses Verfahrens wird weiter unten genauer erklärt. Einer der weiteren Vorteile des GPS-Systems ist neben der hohen Genauigkeit auch dessen breite Verfügbarkeit. Zum einen ist ein einfaches GPS-Empfängergerät heute preiswert zu erwerben und benötigt nur wenig Platz, daher ist GPS von jedermann fast überall einsetzbar. Zum anderen ist die Positionsbestimmung mit GPS nicht von Bedingungen wie der Tageszeit oder der eigenen Position abhängig. Weiteren Auftrieb wird die Verbreitung von GPS aus dem Verzicht des DoD auf die sogenannte *Selective Availability,* einer künstlichen Abschwächung der Radiosignale, die die Positionsgenauigkeit einfacher ziviler GPS-Geräte auf 50 – 200 Meter reduzierte. Diese, abkürzend mit *SA* bezeichnete Maßnahme wurde eingeführt, da potentiellen militärischen Gegnern der USA nicht die vom US-Militär erzielte Positionsgenauigkeit gewährt werden sollte, als Beispiel kann man hier eine durch die Benutzung von GPS in ihrer Zielgenauigkeit verbesserte Langstreckenrakete eines Gegners der USA anführen. Nach dem Ende des kalten Krieges wurden solche Maßnahmen weitgehend überflüssig, was schließlich zur Abschaltung der *SA* am 2. Mai 2000 führte, die wiederum zu einem deutlichen Sinken des relativen Durchschnittsfehlers einer Positionsbestimmung führte. Das DoD behält es sich allerdings vor, die *SA* bei Bedarf wieder einzuführen. Derartig hohe Genauigkeiten konnten bisher nur mit Hilfe des *Differential GPS*-Systems (kurz *DGPS*), einer Verbesserung des einfachen GPS-Systems, erzielt werden. Unabhängig von dieser Verbesserung wird GPS in zwei Varianten angeboten, dem sogenannten *SPS* (**S**tandard **P**ositioning **S**ervice), der jedermann zur Verfügung steht und dem sogenannten, *PPS* (**P**recise **P**ositioning **S**ervice), der vor allem dem US-Militär vorenthalten ist und gegenüber dem *SPS* deutliche Genauigkeitsvorteile bietet.
GPS wird heute von einer Vielzahl von Benutzern in vielen Anwendungsgebieten verwendet. Die Anwendungsgebiete reichen dabei von relativ nahe liegenden, wie z.B. Flugzeug- oder Schiffsnavigation, bis hin zu Anwendungen, die nicht ohne weiteres in Verbindung mit GPS gebracht werden könnten. Die Leistungsfähigkeit, die Einsatzmöglichkeiten und das Funktionsprinzip von GPS und DGPS wird in den folgenden Unterkapiteln diskutiert,

außerdem wird die starke Verbesserung der Positionsgenauigkeit, die sich aus den Wegfall der *SA* ergab noch einmal angesprochen und mögliche Folgen dargestellt.

2.3 Funktionsweise von GPS

Die Funktionsweise von GPS läßt sich grob in vier zusammenhängende Themengebiete unterteilen, jedes dieser Gebiete wird im folgenden noch detailliert erklärt.

1. Die Basis der Positionsbestimmung mit GPS ist die sogenannte *Triangulierung* mit Satelliten, Kapitel 2.3.1 erklärt die Funktionsweise dieses Verfahrens.
2. Um die *Triangulierung* präzise durchführen zu können, bestimmt ein GPS-Empfänger den Abstand zu mindestens vier Satelliten, indem er die Signallaufzeit der Radiosignale von den Satelliten zu ihm selbst mißt. Dieses Vorgehen wird in Kapitel 2.3.2 erläutert.
3. Um diese Reisezeit genau bestimmen zu können, braucht der Empfänger eine äußerst genau gehende Uhr. Erreicht wird dies durch eine genaue Synchronisation der Uhr des Empfängers mit der atomgenauen Uhr eines Satelliten. Das Verfahren, das zur Synchronisation verwendet wird, wird in Kapitel 2.3.3 angegeben und erklärt.
4. Zusätzlich zur Laufzeit der Signale muß der Empfänger jederzeit die Position der einzelnen bei der *Triangulierung* beteiligten Satelliten kennen. Dies wird u.a. erreicht, indem die Satelliten eine sehr hohe, genau vorbestimmte Flugbahn einhalten und auf dieser genau verfolgt werden. Eventuelle Abweichungen von den vorbestimmten Flugbahnen werden den Empfängern über Korrektursignale mitgeteilt. Da alle Signale durch die Atmosphäre gesendet werden, die diese stören und verzögern können, müssen diese Störungen später korrigiert werden, um keine inakzeptablen Positionsfehler zu erhalten. Die Bestimmung der genauen Position und die möglichen Fehlerarten, die diese beeinflussen können, ihre Erkennung und Korrektur werden abschließend in Kapitel 2.3.4 beschrieben.

2.3.1 Triangulierung mit Satelliten

Das Grundprinzip des dem GPS-System zugrundeliegenden Verfahrens zur Positionsbestimmung ist die Messung der Entfernung vom Empfänger zu mindestens vier Satelliten. Werden diese Messungen sehr genau durchgeführt, ist es möglich aus den erhaltenen Werten den Standpunkt des Empfängers äußerst präzise zu errechnen. Dabei steigt die erzielte Positionsschärfe mit steigender Anzahl der Messungen. Zur Erklärung dieses Verfahrens ist es zunächst einmal nicht von Bedeutung, auf welche Weise diese Messungen durchgeführt werden. Dies wird im nächsten Unterkapitel erklärt.
Das GPS zugrunde liegende Verfahren läuft nun folgendermaßen ab: Bestimmt man den Abstand zu einem Satelliten, dann ist dadurch der eigene Aufenthaltsort als Punkt auf der Oberfläche einer Kugel um diesen Satelliten mit einem Radius, der dem bestimmten Abstand gleicht, festgelegt. Durch Messung des Abstands zu zwei Satelliten wird die Menge der möglichen Aufenthaltsorte auf den Schnittkreis zweier Kugeln um die Satelliten eingeengt, deren Radius jeweils entsprechend des bestimmten Abstands ist. Durch Messung des Abstands von drei Satelliten, ist die eigene Position schließlich auf 2 Alternativen genau bestimmt, diese Punkte ergeben sich durch den Schnitt des vorher bestimmten Schnittkreises mit der Kugel um den dritten Satelliten. Oft ist es möglich, einen dieser beiden gefundenen möglichen Aufenthaltspunkte auszuschließen, da sich eine der beiden Positionen nicht auf oder nahe bei der Erde befinden wird, sondern weit entfernt im All. Trotzdem wird aus Gründen der Fehlerbestimmung noch mindestens eine zusätzliche Abstandsbestimmung zu

einem Satelliten benötigt. Dies wird in Kapitel 2.3.3 noch genauer erklärt. Der Inhalt dieses Kapitels soll nun noch mit einem Beispiel verdeutlicht werden: Angenommen, die Entfernung vom Empfänger zu einem bestimmten Satelliten wurde durch Messung auf 21.000 Kilometer bestimmt. Das Wissen, 21.000 Kilometer von einem bestimmten Satelliten entfernt zu sein, engt die Menge der möglichen Aufenthaltspunkte auf die Oberfläche einer Kugel um den Satelliten mit Radius 21.000 Kilometer ein. Dies wird in Abbildung 1 illustriert.

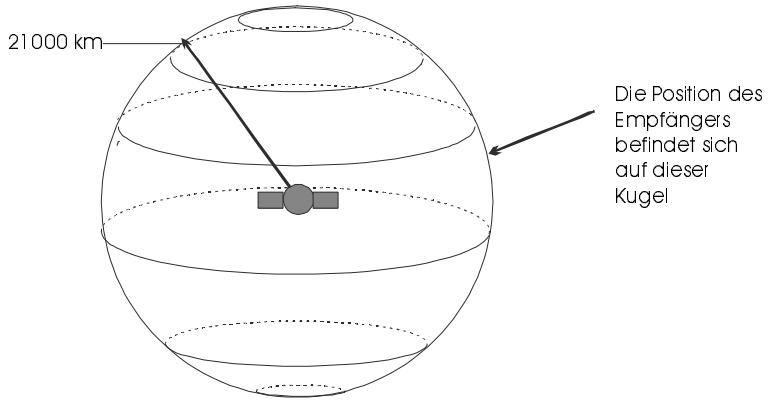


Abbildung 1 Mögliche Aufenthaltspunkte bei Messung des Abstands von einem Satelliten

Als nächstes bestimmt man die Entfernung zu einem anderen Satelliten zu 22.000 und die Entfernung zu einem dritten zu 23.000 Kilometern.
Dies hat als Folge, daß die zu bestimmende Position einer der beiden Punkte sein muß, die sich durch den Schnitt aller drei „Abstandskugeln“ ergeben. Einer dieser beiden Punkte liegt in der Nähe oder auf der Erde, der andere weit entfernt im All. Dementsprechend kann eine der beiden Lösungen vernachlässigt werden und die eigene Position ist damit bekannt.

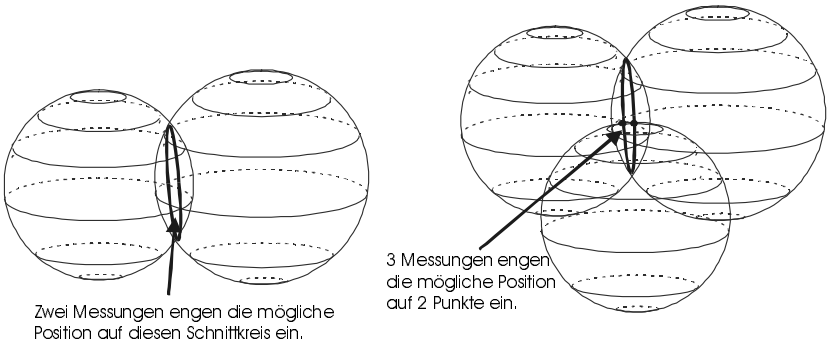


Abbildung 2: Mögliche Aufenthaltspunkte bei Messung des Abstands von zwei und drei Satelliten

2.3.2 Abstandsbestimmung zu Satelliten

Im vorherigen Unterkapitel wurde das Verfahren, das GPS zugrunde liegt erklärt, nämlich wie aus bekannten Abständen zu Satelliten der aktuellen Aufenthaltspunkt bestimmbar ist. In diesem Unterkapitel wird nun besprochen, auf welche Weise sich der Abstand zu einem Satelliten bestimmen läßt.

Die Grundidee zur Abstandsbestimmung ist folgende:
Der Abstand läßt sich berechnen aus der Größe der Laufzeit, die ein Signal vom Satelliten zum Empfänger benötigt, und der Größe der Geschwindigkeit die ein solches Signal innehat. Außerdem wird die Geschwindigkeit des Signals als konstant angenommen, d.h. man vernachlässigt eventuelle Beschleunigungs- bzw. Bremsvorgänge. Dann läßt sich die Frage nach dem Abstand zurückführen auf eine einfache Multiplikation:

*Abstand vom Satelliten = Laufzeit des Signals * Geschwindigkeit des Signals*

Dabei ist die Geschwindigkeit des Signals bekannt, denn es ist ein Radiosignal und somit gleicht dessen Geschwindigkeit der Lichtgeschwindigkeit, also ungefähr 340.000 km/s. Es bleibt noch die Frage zu klären, wie die Reisezeit bestimmt wird. Die Zeitmessung muß dabei äußerst präzise sein, denn bereits kleine Abweichungen werden durch die große Geschwindigkeit zu großen Entfernungsfehlern aufmultipliziert. So kann z.B. eine Zeitfehler von 0,1 µs bereits zu einem Positionierungsfehler von 30 m führen. Gemessen wird die Zeitspanne, die sich vom Zeitpunkt des Erzeugens und Absendens eines Signals durch einen Satelliten bis zum Empfang durch den GPS-Empfänger ergibt. Ein von einem Satelliten emittiertes Signal besteht eigentlich aus zwei Trägern, deren im Mikrowellenbereich befindlichen Frequenzen für alle Satelliten gleich sind. Da so keine Zuordnung eines empfangenen Signals zu einem bestimmten Satelliten erfolgen kann, werden zur Unterscheidung bestimmte Codes auf die Träger aufmoduliert. Diese werden als Pseudo Random Code (PRC) bezeichnet. Ein PRC ist ein relativ komplexer digitaler Code, oder eine relativ zufällig aussehende Abfolge von elektrischen 0- und 1-Signalen, weshalb man ihn auch als „Pseudo-Random“ bezeichnet.

Abbildung 3: Pseudo Random Code[1]

Jeder Satellit hat seinen eigenen Pseudo Random Code, mit dem Vorteil, daß der Empfänger jedes Signal genau einem bestimmten Satelliten zuordnen kann. Der Vorteil der Komplexität wiederum ist, daß es äußerst unwahrscheinlich ist, daß ein Empfänger ein zufällig von irgendeinem anderen Objekt erzeugtes Signal mit einem Signal eines Satelliten verwechseln kann. Auf diese Weise wird Fehlern, die kritische Folgen haben könnten, vorgebeugt und alle Satelliten können auf derselben Frequenz senden, denn der Absender steht durch den aufmodulierten PRC des Signals fest. Aber es gibt noch einige andere Vorteile, die die komplizierte Struktur der PRC beinhalten. So gilt z.B.: Je komplexer das Signal ist, desto schwerer ist es zu stören bzw. zu verfälschen. Außerdem macht es diese Art von Code möglich, das Signal zu verstärken, mit dem Vorteil, daß es auf diese Weise auch von kleinen Empfängern mit kleinen Antennen empfangen werden kann. Dieser Vorteil trug nicht unwesentlich zu der weiten Verbreitung des GPS-Systems bei. Es werden die zwei folgenden Klassen von PRC für die Trägersignale jedes Satelliten verwendet:

- C/A-Code (coarse/acquisition code, d.h. Grob/Auffaßcode). Wird auf den ersten Träger aufmoduliert für den SPS (Standard Positioning Service).
- P-Code (precison code). Wird auf beiden Trägern aufmoduliert und für den PPS (Precise Positioning Service) genutzt.

Die Unterschiede zwischen diesen beiden Codearten bestehen z.B. in der Taktfrequenz, die beim P-Code um den Faktor 10 höher liegt als beim C/A-Code. Dies hat zur Folge, daß durch Verwendung des P-Codes eine höhere zu erhaltende Positionsschärfe möglich ist. Genaueres dieses Thema betreffend ist z.B. in [DOC96] zu erfahren. Dem auf die beiden Träger

[1] Abbildung entnommen aus [TRI97]

aufmodulierten PRC wird zur eigentlichen Informationsübermittlung selbst ein Bitstrom aufmoduliert, dieser wird oft mit „NAV-Nachricht“ bezeichnet. Auf diese Weise können durch Übertragung eines Signals von einem Satelliten zu einem Empfänger u.a. folgende Informationen übermittelt werden:

- Die Position des Satelliten zum Sendezeitpunkt des Signals,
- eine Zeitmarke, die den Sendezeitpunkt angibt,
- die Bahndaten des Satelliten,
- Korrekturwerte für die Verzögerung des Signals durch Atmosphäreneinflüsse (siehe hierzu 2.3.4),
- Daten für die Positionsbestimmung der einzelnen Satelliten (Almanache, siehe 2.3.4).

Der eigentliche Vorgang der Zeitmessung kann folgendermaßen beschrieben werden:
- Der Empfänger vergleicht nach Empfang eines Signals eines Satelliten die in diesem Signal enthaltene Zeitmarke des Signals mit dem Empfangszeitpunkt des Signals.
- Die Differenz dieser beiden Werte ist dann die Laufzeit des Signals.

Hier wird wieder die Wichtigkeit der genauen Uhrensynchronisation von Empfänger- und Satellitenuhr deutlich, das dazu verwendete Verfahren wird im nächsten Kapitel erklärt.

2.3.3 Synchronisation zwischen GPS-Empfängeruhr und Satellitenuhr

Zusätzlich zur Synchronisation der Satelliten– und Empfängeruhren ist für die Anwendbarkeit des GPS zugrunde liegenden Verfahrens zur Positionsbestimmung auch die Verwendung möglichst genau gehender Uhren von großer Bedeutung. Zu den genauesten Uhren gehört die Atomuhr. Und in der Tat hat jeder der 24 GPS-Satelliten eine eigene eingebaute Atomuhr. Das Verwenden eigener Atomuhren in den Empfängern ist allerdings nicht praktikabel, denn Atomuhren sind äußerst teuer (50.000 – 100.000 US-Dollar). Ein Einbau einer Atomuhr in jeden Empfänger würde also auch dessen Preis um diese Summe anheben, was natürlich völlig indiskutabel war. Das bei GPS verwendete Verfahren ist allerdings in der Lage, einer Empfängeruhr praktisch die Genauigkeit einer Atomuhr zu verleihen. Diese Verbesserung der Uhrengenauigkeit wird durch die Durchführung einer zusätzlichen Messung des Abstands zu einem Satelliten erreicht. Wie oben bereits erwähnt, reichen für die Bestimmung der Position in einem dreidimensionalen Raum drei Abstandsmessungen zu Punkten bekannter Position aus. Durch das Durchführen einer zusätzlichen Abstandsmessung läßt sich nun vom Empfänger ein Korrekturfaktor bestimmen, mit dessen Hilfe Meßungenauigkeiten, die durch die ungenaue Empfängeruhr verursacht wurden, ausgeglichen werden können. Der Korrekturfaktor läßt sich bei n durchgeführten Messungen durch Vergleich der mit jeweils n-1 Satelliten bestimmten Positionen errechnen. Man erreicht also bei diesem Verfahren durch das Durchführen einer zusätzliche Messung eine Überbestimmung der eigenen Position, aus der sich wiederum die die Uhrenabweichung bestimmen läßt. Die Bestimmung des Korrekturfaktors soll im folgenden an einem Beispiel verdeutlicht werden, das der Übersichtlichkeit halber nur in einem zweidimensionalen Raum stattfinden wird.
Angenommen, die eigene Position ist 4 Sekunden von Satellit A und 6 Sekunden von Satellit B entfernt. Allerdings wird durch die ungenau gehende Empfängeruhr der zeitliche Abstand zu Satellit A auf 5 Sekunden und zu Satellit B auf 7 Sekunden bestimmt. Die durch diese Meßfehler entstehende Situation ist in Abbildung 4 dargestellt.

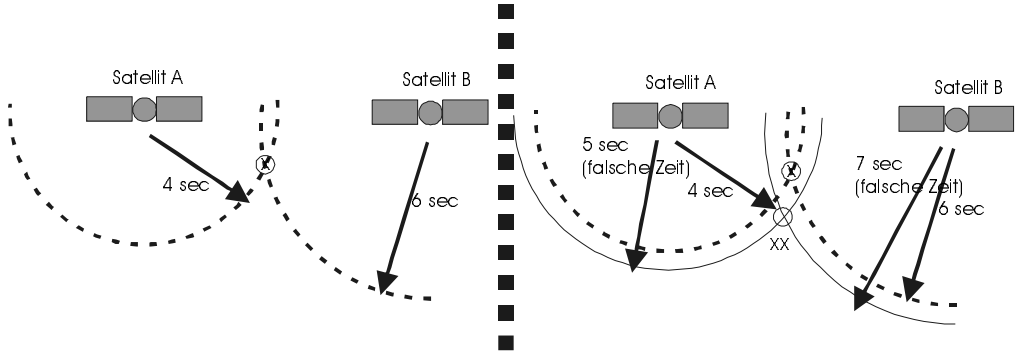


Abbildung 4: Entstehender Positionsfehler durch ungenaue Empfängeruhr

Die in Abbildung 4 durchgehend gezeichneten Kreise zeigen die durch die ungenaue Uhr erzeugten *Pseudo-Reichweiten (*engl.: *Pseudo Ranges).* Der Begriff *Pseudo-Reichweite* wird gewöhnlich immer dann verwendet, wenn Messungen fehlerbehaftet sein können. Bei Durchführung einer weiteren Messung wurde dann die *Pseudo-Reichweite* zu einem Satelliten C auf 9 Sekunden bestimmt, der korrekte Zeitabstand betrug hier 8 Sekunden. Der GPS-Empfänger kann nun anhand der drei *Pseudo-Reichweiten* erkennen, daß eine Abweichung der von ihm bestimmten Werte zu den korrekten Werten, die bei einer perfekten Empfängeruhr ermittelt worden wären, besteht. Dies wird in Abbildung 5 illustriert.

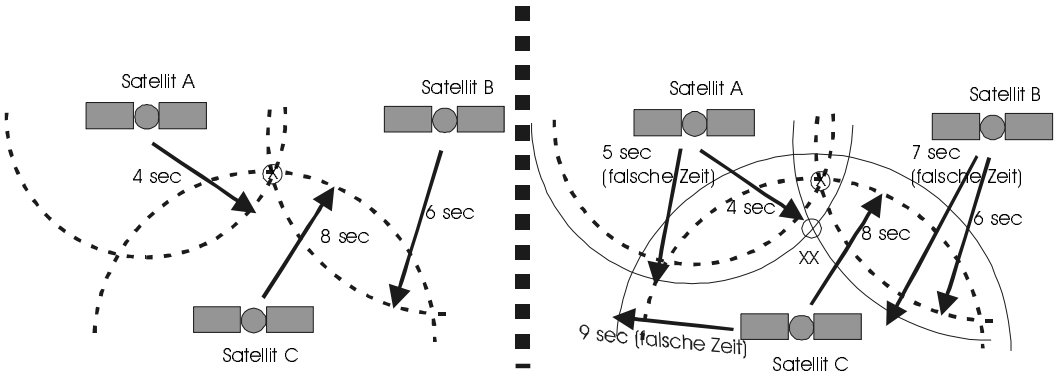


Abbildung 5: Erkennen einer Uhrendrift durch Durchführen einer zusätzlichen Messung

Dies ist der Fall, weil sich aus geometrischen Gründen nicht alle drei *Pseudo-Reichweiten* in einem Punkt schneiden können. So wird in Abbildung 5 deutlich, daß sich z.B. die beiden *Pseudo-Reichweiten* der Satelliten A und B im Punkt XX schneiden, aber die *Pseudo-Reichweite* um Satellit C nicht durch diesen Punkt verläuft. Der Empfänger kann daran erkennen, daß seine Uhr ungenau ist und kann nun aus den erhaltenen Werten einen Korrekturwert bestimmen, durch den sich alle 3 *Pseudo-Reichweiten* in einem Punkt schneiden würden. In diesem Beispiel würde der Empfänger den Korrekturwert zu -1 Sekunde bestimmen, d.h. ein gemeinsamer Schnittpunkt würde entstehen, wenn bei allen Zeitabstandsmessungen eine Sekunde abgezogen werden würde. Dieser Korrekturwert kann nun auf alle folgenden Messungen angewendet werden und damit kann man die Uhr des Empfängers als annähernd atomgenau ansehen. Diese Bestimmung eines Korrekturwerts wird im weiteren Verlauf einer Positionsbestimmung in regelmäßigen Abständen wiederholt, damit eventuelle Veränderungen des Werts berücksichtigt werden können. Genaueres zur Bestimmung des Korrekturwerts kann z.B. in [NEX99] eingesehen werden.

Das oben erklärte Verfahren zur Uhrensynchronisation hat unter anderem den Nebeneffekt, daß ein GPS-Empfänger außer dem offensichtlichen Verwendungszweck der Positionsbestimmung außerdem noch als atomgenaue Masteruhr eingesetzt werden kann. Mögliche Anwendungen wären z.B. die Synchronisation der Uhren in verteilten Systemen oder die Kalibrierung anderer Navigationssysteme.

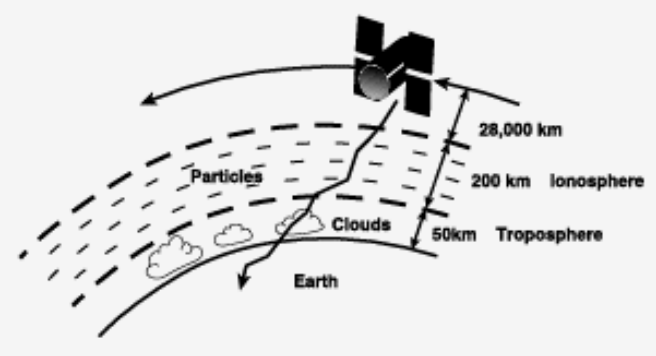
Weiterhin hat dieses Verfahren für einen GPS-Empfänger zur Konsequenz, daß er mindestens vier Kanäle anbieten muß, über die die einzelnen Messungen parallel durchgeführt werden können. Ohne dieses Verfahren würden bereits drei Kanäle ausreichen (siehe 2.3.1). Heutige aktuelle Empfänger verfügen allerdings bereits über 12 Kanäle, so daß diese Forderung ohnehin schon erfüllt ist.

Außer einer korrekten Bestimmung des Abstands zu bestimmten Satelliten muß zur *Triangulierung* einer Position aber auch noch der genaue Aufenthaltspunkt dieser Satelliten bekannt sein.

2.3.4 Mögliche Fehlerquellen

Die Grundlage für eine erfolgreiche Positionsbestimmung der einzelnen Satelliten liegt im sogenannten *GPS-Masterplan.* Dieser wurde vom DoD bei der Planung des GPS-Systems entworfen und von der US-Airforce bei der Installation der Satelliten im Orbit umgesetzt.
Im *GPS-Masterplan* wird u.a. festgelegt, daß die Satellitenkonstellation so beschaffen sein muß, daß auf jedem Punkt der Erde zu jeder Zeit mindestens 4 Satelliten gleichzeitig „sichtbar“ sein müssen. Dies ist eine der Voraussetzungen, die für den Erhalt einer korrekten Position notwendig sind (siehe hierzu 2.3.3).
Nachdem die Position der einzelnen Satelliten durch den *Masterplan* festgelegt wurde, können die GPS-Empfänger anhand ihrer *Almanache* feststellen, wo sich ein bestimmter Satellit wann befindet. Unter *Almanach* wird hier eine Datensammlung verstanden, die z.B. die Bahndaten aller Satelliten enthält. Kurz gesagt, sämtliche Daten, die für eine Positionsbestimmung von Satelliten notwendig sind, werden in solchen *Almanachen* beschrieben. Beim Neustart eines GPS-Empfängers muß dieser zuerst überprüfen, ob der gespeicherte *Almanach* noch aktuell ist, oder bereits veraltet ist. Ein *Almanach* kann bis zu 180 Tage gültig sein, nach dieser Zeit weichen die hier gespeicherten Orbitaldaten der einzelnen Satelliten von den tatsächlichen Orbitaldaten um ein unzulässiges Maß ab und es muß ein aktueller *Almanach* von einem Satelliten bezogen werden. Moderne Empfänger beziehen automatisch den jeweils aktuellen *Almanach* von den Satelliten. Der Transfer wird, wie bereits angesprochen, mittels der „NAV-Nachricht“, die auf jedes Satellitensignal aufmoduliert ist, durchgeführt. Um eventuelle temporär auftretende Abweichungen vom *Almanach,* die z.B. durch die Anziehungskraft des Mondes und der Sonne oder durch den Einfluß der Sonnenstrahlung auf die Satelliten entstehen können, auszugleichen werden die Bahndaten der Satelliten durchgehend vom DoD überwacht. Man spricht bei diesen Fehlern von *Ephemeresfehlern*, da sie den Orbit (= Ephemeres) eines Satelliten betreffen. Es existieren hierzu spezielle Bodenstationen, die die aktuelle Position der „sichtbaren“ Satelliten mit Hilfe einer Hochpräzisionsradaranlage bestimmen und mit den im *Almanach* eingetragenen Werten vergleichen. Bei signifikanten Abweichung der beiden Werte wird der bestimmte Fehler in einer Nachricht an die betroffenen Satelliten gesendet. Diese strahlen die Korrekturdaten, zusammen mit den *PRCs (*siehe hierzu 2.3.2) auf die Signalträger aufmoduliert, wieder aus. GPS-Empfänger, die ihre Position durch diese Satelliten bestimmen, können dann während des Betriebs anhand der Korrekturdaten ihren *Almanach* lokal korrigieren und damit präzise Positionen erhalten. Eine Neubestimmung der Korrekturdaten ist für jeden Satelliten mindestens alle 4 Stunden erforderlich. Doch es gibt

außer den *Epheneresfehlern* noch weitere Fehlerquellen, die eine exakte Positionsbestimmung verhindern können. Dabei ist vor allem die Konstellation des GPS-Systems mit im Orbit befindlichen Satelliten und auf oder in der Nähe der Erde befindlichen Empfängern die größte potentielle Fehlerquelle. Durch diese Konstellation ist es unvermeidlich, daß Signale vom Satelliten zum Empfänger durch die Atmosphäre gesendet werden müssen. Dadurch ist relativ wahrscheinlich, daß sie dort verzögert oder gestört werden. Die oberste, die Signale beeinflussende Schicht der Atmosphäre ist die Ionosphäre. In dieser Schicht, die sich 50 – 500 km über der Erdoberfläche befindet, befinden sich vor allem ionisierte (= elektrisch nicht neutrale) Partikel. Diese können den Verlauf eines Signals stark beeinflussen und stellen eine der größten potentiellen Fehlerquellen für ein Signal dar. Darunter befindet sich die Troposphäre, die vor allem aus Wasserdampf besteht. Diese Wassertröpfchen können ein Signal ebenfalls stören, allerdings nicht in dem Maße wie die Ionosphäre.



.

Abbildung 6: Weg eines Signals durch die Erdatmosphäre[2]

Um diese Fehler zu beseitigen, sind zwei Ansätze gefunden worden, die auch zusammen verwendet werden können.

- Erstellung eines mathematischen Modells der Atmosphäre.
  Mit diesem wird versucht, Vorhersagen über die Ablenkung der Signale zu erstellen. Ein Modell setzt die einzelnen Schichten der Atmosphäre in Relation zu den potentiellen Fehlern, die sie verursachen können. Nach Erstellung eines solchen Modells kann in Abhängigkeit des Eintrittswinkels des Signals in die Atmosphäre die ungefähre Größe des verursachten Fehlers bestimmt werden.
- Bestimmung des Verzögerungswerts mittels einer sogenannten „Doppel-Frequenzmessung“
  Darunter wird ein Verfahren verstanden, bei dem anhand eines Vergleichs zweier Signale unterschiedlicher Frequenz und ihrer jeweiligen Verzögerung nach Durchqueren der Atmosphäre auf den Typ des Fehlerverursachers und damit auf den verursachten Fehler rückgeschlossen wird.

Eine weiteres Fehlerpotential wird beim Eintreffen des Signals an der Erdoberfläche deutlich. So kann es passieren, daß ein Signal von sich auf der Erdoberfläche befindlichen Hindernissen abgelenkt oder verzögert wird, wie z.B. von Bäumen oder Häusern. Dieser Fehler wird *Multipath*-Fehler genannt, weil ein Empfänger in diesem Fall typischerweise zuerst das direkte Signal vom Satelliten empfängt und nach einer gewissen Verzögerung mehrere zuvor abgelenkte und damit verzögerte Signale. Sind diese Signale hinreichend stark, kann es zu Fehlern bei der Signalauswertung und damit zu Positionierungsfehlern kommen. Moderne Empfänger haben allerdings Möglichkeiten, verzögerte Signale zu ermitteln. Der

[2] Abbildung entnommen aus [TRI97]

Multipath-Fehler kann durch die Wahl des Standortes zur Positionsbestimmung stark gesenkt werden. Abbildung 7 stellt das Entstehen eines *Multipath*-Fehlers dar.

Abbildung 7 : Enstehung des Multipath-Fehlers[3]

Zum Abschluß dieses Unterkapitels soll nun noch auf einen geometrischen Effekt eingegangen werden, der übergreifend als *Geometric Dilution of Position* (kurz: *GDOP*) bezeichnet wird: Ein GPS-Empfänger kann normalerweise aus einer Anzahl von Satelliten zur Positionsbestimmung auswählen. Wählt der Empfänger aus den zur Verfügung stehenden Satelliten eine Menge von Satelliten aus, deren Positionen relativ nahe beisammen liegen, dann schneiden sich die Standlinien, die die Position des Empfängers festlegen, in einem relativ flachen Winkel. Dies bedeutet, daß sich die Grauzone, in der die tatsächliche Position liegt, vergrößert. Wählt der Empfänger dagegen Satelliten aus, deren Aufenthaltsorte weit voneinander entfernt sind, dann schneiden sich die Standlinien fast rechtwinklig und damit wird die Größe der Grauzone minimiert. Dies wird in Abbildung 8 veranschaulicht.

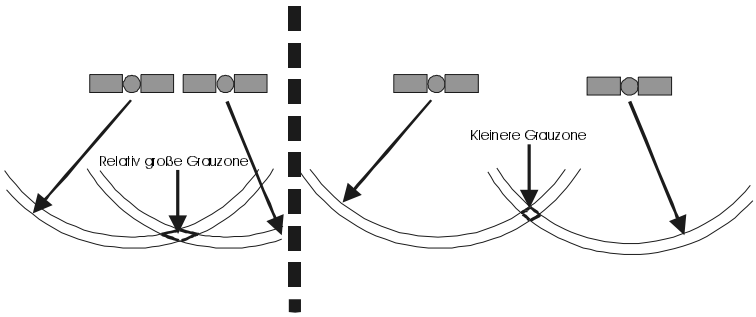


Abbildung 8: Einfluß der Satellitenauswahl auf die erzielte Positionsschärfe

Eine letzte potentielle Fehlerquelle ist wie zuvor bereits angesprochen, seit 2. Mai 2000 nicht mehr aktuell. Es handelt sich dabei um die absichtliche Signalverschlechterung, die vom DoD eingeführt wurde, um potentiellen Gegnern der USA und ihrer Verbündeten nicht die gleiche Möglichkeit bei der Positionsbestimmung zu bieten wie dem US-Militär. Dies wurde z.B. durch künstliche Ungenauigkeiten in den Satellitenuhren, oder durch kleine Veränderungen in den Ephemeresdaten der Satelliten erzielt, also durch Maßnahmen, die zu einer Verschlechterung der Positionsschärfe führen. Diese Maßnahmen wurden zusammengefaßt unter dem Begriff *Selective Availability*. Sie verschlechterte die Positionsschärfe ungefähr um den Faktor 10. Um trotzdem eine genauere Position bestimmen zu können, wurde daraufhin

[3] Entnommen aus [TRIL97]

das DGPS-Verfahren (Differential GPS) entwickelt, das mit Hilfe neuer Korrekturnachrichten diesen Nachteil wieder ausgleichen konnte. Trotz des Abschaltens der *Selective Availability* ist das DGPS-Verfahren immer noch aktuell, denn es ist in der Lage auch das nun viel genauere einfache GPS-Verfahren zu verbessern. Im nächsten Abschnitt wird genauer auf DGPS eingegangen werden.

Um den relativen Anteil dieser verschiedenen Fehlerquellen am Gesamtfehler abschätzen zu können, wird abschließend noch ein Auszug[4] aus einer Tabelle, die ein „Fehlerbudget" für die einzelnen Fehlerarten bei einer 24-stündigen Messung enthält, angegeben. Die dabei benutzte sogenannte „95%-UERE"-Metrik (**U**ser **E**quivalent **R**ange **E**rror, d.h. Benutzeräquivalenter Abstandsfehler) ist ein zeitabhängiges Maß, mit dem der empfängerseitig bestimmbare Fehler in der Abstandsmessung zu jedem Satelliten eingeschätzt werden kann. Die dabei angegebene Prozentzahl definiert die angegebenen Zahlenwerte als die Fehlerwerte, unter deren Betrag 95% aller bestimmten absoluten Fehlerwerte lagen. Zusätzlich wird noch zwischen *PPS* und *SPS* unterschieden.

Fehlerquelle	**95%-UERE für P-Code [m]**	**95%-UERE für C/A-Code [m]**
Verzögerung in Ionosphäre	4,5	9,8 – 19,6
Verzögerung in Troposphäre	3,9	3,9
Multipath-Fehler	2,4	2,4
Andere Fehler	1,0	1,0

Tabelle 1: 95%-UERE für ausgesuchte Fehlerarten

2.4 DGPS

Differential GPS wurde entwickelt, als das einfache GPS-System für den gewöhnlichen Benutzer durch die *Selective Availability* noch künstlich ungenau gehalten wurde. Es mußten Wege gefunden werden, trotzdem eine größere Positionsschärfe erzielen zu können, was für einige potentielle Anwendungsgebiete des GPS-Systems, wie z.B. der Landvermessung, eine Grundvoraussetzung war. Das Grundkonzept, das DGPS verfolgt ist das folgende: Um die künstlichen und natürlichen Fehler, die im Vorgang der Positionsbestimmung auftreten, zu minimieren, existieren Referenzempfänger genau bekannter Position, die in der Lage sind, den von den im vorigen Abschnitt beschriebenen verschiedenen Fehlerquellen aufsummierten Fehler zu bestimmen und diesen an die umliegenden GPS-Empfänger weiterzugeben. Diese können mit der Fehlerinformation Maßnahmen zur Korrektur einleiten. Möglich ist dieses Verfahren, da die Flughöhe der Satelliten im All, im Gegensatz zum Abstand der Empfänger vom Referenzempfänger, sehr groß ist. Dadurch ist der summierte Fehler bei Empfängern, die sich innerhalb eines gewissen Radius zum Referenzempfänger befinden, praktisch gleich dem des Referenzempfängers, da die empfangenen Signale ja alle den nahezu gleichen Transferbedingungen ausgesetzt waren. Durch das DGPS Verfahren können also die durch Signalverzögerungen in der Atmosphäre hervorgerufenen Positionsfehler beseitigt werden, nicht beseitigt werden können mit diesem Verfahren lokal auftretende Fehler, wie der *Multipath*-Fehler, der erst lokal um einen Empfänger herum entsteht. Die Funktionsweise des DGPS-Systems läßt sich nun folgendermaßen beschreiben:

- Die Referenzempfänger wurden an festen Punkten mit genau bestimmten Koordinaten installiert.

[4] Tabelle aus [DOC96] entnommen

- Ein Referenzempfänger berechnet, wie lange ein Signal von den jeweiligen Satelliten brauchen dürfte.
- Die Referenzdauer und die aktuelle Signaldauer werden verglichen und die Differenz bestimmt.
- Aus dem Unterschied läßt sich ein Korrekturfaktor bestimmen, der für alle GPS-Empfänger im Umkreis von etwa 200 bis 250 km gilt. Dieser Wert bestimmt sich als Grenzwert der Entfernung vom Referenzempfänger, für den für diesen und einen mobilen Empfänger noch dieselben Signalverzögerungen angenommen werden können, denn je weiter sich ein mobiler Empfänger vom Standpunkt eines Referenzempfängers entfernt, desto mehr unterscheiden sich die Eintrittswinkel der Signale in die Atmosphäre vom Satelliten zu den beiden Empfängern. Da vom Eintrittswinkel die Dauer des Aufenthalts in der Atmosphäre und davon wiederum die Wahrscheinlichkeit einer Signalverzögerung abhängt, wurde oben genannter Grenzwert festgelegt. Größere Distanzen eines mobilen Empfängers zu einer Referenzstation sind möglich, allerdings geht dies auf Kosten des Verbesserungspotentials, das das DGPS-System gegenüber dem einfachen GPS-System bietet.

Da ein Referenzempfänger nicht feststellen kann, welche der möglichen Satelliten die einzelnen Empfänger zur Bestimmung der Position benutzen, bestimmt er für alle ihm sichtbaren Satelliten den Korrekturfaktor und sendet diese Liste an die Empfänger, die daraus die von ihnen verwendeten Satelliten und den zugehörigen Korrekturfaktor entnehmen. Die Übermittlung findet dabei meist über ein Langwellensignal, üblicherweise im Bereich um 300kHz statt. Die Korrektursignale sind dabei gemäß dem RTCM-Format aufgebaut (RTCM – Radio Technical Commission for Maritime Services). Es existieren noch andere Arten der Signalübertragung, z.B. über UKW durch Radiostationen mittels des bekannten RDS-Systems, parallel zu deren herkömmlichen Rundfunkprogrammen. Die Reichweite der Signale ist hier allerdings auf 60 bis 100 Kilometer beschränkt. Eine weitere Möglichkeit wäre auch die Nutzung des Internets zur Signalübertragung. Die Überdeckung der Erdoberfläche mit Referenzempfängern nimmt stetig zu, vor allem im Bereich großer Häfen oder Wasserstraßen.

DGPS ermöglichte bei eingeschalteter *Selective Availability* die Position ca. um den Faktor 10 genauer bestimmen zu können. Dies führte zu einer Positionsschärfe von ungefähr 5 Metern. Auch nach dem Abschalten der *SA* führt DGPS noch zu einer Verbesserung der Positionsschärfe, allerdings nicht mehr in dieser Größe. Tabelle 1 wurde aus [MIL00] entnommen und stellt die Auswertung einer über drei Tage laufende Positionsbestimmung eines GPS-Empfängers und eines DGPS-Empfängers nach dem 2.5.2000 dar. Mit RMS-Fehler (***R**oot **M**ean **S**quare*- Fehler) ist dabei der Wert gemeint, der sich durch folgende Formel errechnet:

$$\text{RMS error} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|p_i - \hat{p}_i\|^2},$$

Hierbei bezeichnet *p^* eine durch einen GPS-Empfänger bestimmte, möglicherweise fehlerhafte Position, während durch *p* eine ohne Fehler bestimmte Position bezeichnet wird. Dies bedeutet, daß bei der Berechnung des RMS-Fehlers die durchschnittliche Abweichung aller bestimmten Positionen eines gewissen Zeitraums von den eigentlich jeweils korrekten Positionen bestimmt wird. Der RMS-Fehler ist eine wichtige Metrik zur Einordnung von Positionsabweichungen. Die den Werten der vierten Zeile zugrunde liegende Metrik wird mit CEP (***C**ircular **E**rror **P**robable)* bezeichnet, sie gibt den Radius des kugelförmigen Bereichs

in Metern an, in dem sich mindestens 50% der bestimmten Positionen befanden. Entsprechend sind die Werte in der fünften Zeile zu verstehen, sie geben den Radius der Kugel an, in der sich 95% der bestimmten Positionen befanden. Im Zusammenhang mit den beiden letztgenannten Fehlermetriken spielt die Gauß-Normalverteilung eine wichtige Rolle.

	Position	**Höhe**	**Position (DGPS)**	**Höhe (DGPS)**
Anzahl der Messungen	129354	129354	128971	128971
Durchschnittlicher Positionsfehler [m]	4,6	11,4	3,4	5,8
RMS-Fehler [m]	5,6	13,9	4,8	9,0
50 % der gemessenen Werte innerhalb Kugel mit Radius [m]	3,8	10,3	2,6	4,1
95 % der gemessenen Werte innerhalb Kugel mit Radius [m]	10,0	25,8	8,2	17,1

Tabelle 2 : Fehlerkennwerte nach Abschalten der SA

Obwohl diese Tabelle mangels Häufigkeit der Messungen nicht als repräsentativ gelten kann, offenbart sie doch ein gewisse Tendenz. Es ist ersichtlich, daß der Vorsprung an Genauigkeit, den DGPS-Empfänger früher gegenüber normalen GPS-Empfängern inne hatten (Faktor 10) nach Abschalten der *SA* relativ stark gesunken, aber immer noch vorhanden ist. Ob dies den deutlich teueren Preis der DGPS-Geräte gegenüber den einfachen GPS-Geräten rechtfertigt, muß bei Kaufabsicht eines neuen Gerätes wohl von Fall zu Fall entschieden werden.

2.5 Mögliche Anwendungen von GPS

Das GPS-System bietet eine überraschend große Anzahl an Verwendungsmöglichkeiten, die weit über den offensichtlichen Zweck der Positionsbestimmung herausgehen. Im folgenden werden einige dieser Anwendungen aufgelistet und kurz beschrieben.

- **Navigation**:
  GPS wird z.B. in der Seefahrt häufig eingesetzt. Die Vorteile liegen auf der Hand, mit GPS wird Navigation zur See erheblich erleichtert, da keine Fixpunkte o.ä. mehr nötig sind, um die Position zu bestimmen. Günstig ist außerdem der verringerte Gesamtfehler durch das Nichtvorhandensein eines *Multipath*-Fehlers (keine Häuser, Bäume usw.). Eine neuseeländische Fischereifirma nutzt GPS, um problemlos zu einmal bestimmten ertragreichen Fischgründen zurückkehren zu können.
  Auch in der Luftfahrt wird GPS vermehrt verwendet. So werden Positionsberechnungen im Flugzeug oft mit auf GPS basierenden Tools durchgeführt und auf häufig beflogenen Flugrouten die Abstände zwischen den Flugzeugen mit GPS bestimmt.
  Zunehmend setzt sich GPS auch bei Wanderern, Tourenradfahrern, Reisenden usw. durch. Gerade für den privaten Einsatz werden von praktisch allen Herstellern sogenannte GPS-Mäuse angeboten, die Handlichkeit und Komfortabilität vereinen. Diese sind leicht, klein, schon preiswert zu erwerben und bieten eine relativ große Anzahl an Funktionen. Oftmals besitzen diese Geräte ein eigenes Display, auf dem z.B. eine Karte eingeblendet, die

gerade bereiste Strecke eingezeichnet oder auch die Richtung zum nächsten Wegpunkt abgelesen werden kann. Bekannt ist auch der Einsatz von GPS im Bereich der elektronischen Navigationssysteme, die in Automobilen verstärkt Verwendung finden.

- **Logistik**:
  GPS eignet sich nicht nur zur Bestimmung der eigenen Position, sondern auch zur Bestimmung des momentanen Aufenthaltsorts z.B. der LKW einer Spedition. Die Logistik dieser Firma würde sich deutlich vereinfachen, denn die Position jedes ihrer LKW wäre zu jedem Zeitpunkt bekannt und es könnten einfach kurzfristige Aufträge abhängig vom momentanen Aufenthaltsort delegiert werden. Eingesetzt wird GPS in diesem Zusammenhang z.B. in einem australischen Taxiunternehmen, das so feststellt, wo sich die einzelnen Taxis im Moment befinden.
- **Kartographie**:
  (D-)GPS ist durch die Fähigkeit, die Position äußerst genau bestimmen zu können, bestens geeignet, um Karten zu erstellen, oder in vorhandene Karten bestimmte Gebiete zu identifizieren. So wird DGPS von Landvermessungstrupps verwendet, um die eigene Position genau bestimmen zu können. Die dabei benutzten Geräte sind allerdings hochpräzise, sehr teuer und nicht mit den üblichen GPS-Empfängern zu vergleichen.
- **Zeitgebung**:
  Da die interne Uhr jedes GPS-Empfängers annähernd atomgenau ist, kann man jedes Gerät auch dazu benutzen, um Uhren miteinander zu synchronisieren. Jedes GPS Gerät ist auch eine äußerst genaue Masteruhr, auf die sich andere Uhren beziehen können.
- **Landwirtschaft**:
  GPS wird teilweise von landwirtschaftlichen Großbetrieben dazu genutzt, eine Überdüngung bestimmter Abschnitte zu vermeiden, indem bereits gedüngte Flächen mit GPS vermessen werden und als bereits besucht gekennzeichnet werden können. Weiterhin kann durch Messung der Ertragssteigerung auf gedüngten Flächen herausgefunden werden, ob sich eine Düngung in diesem Gebiet als rentabel erweist.

2.6 Das NMEA Protokoll

Um die Daten, die der GPS-Empfänger bestimmt hat, für sich nutzbar zu machen, muß ein Protokoll für die Übertragung der Daten zwischen GPS-Gerät und Computer, sowie für die Steuerung des GPS-Geräts vom Computer aus, bereitgestellt werden. Ein solches ist das NMEA 0183 Protokoll, ein von der *National Marine Electronics Association* entworfenes Standardprotokoll für die Kommunikation über eine serielle (RS 232-kompatible) Schnittstelle.

Die Ursprünge dieses Protokolls liegen in dem im Jahre 1980 ebenfalls von der *National Marine Electronics Association* entworfenen NMEA 0180 Standard, der für das Zusammenarbeiten des LORAN-Systems mit Autopiloten entworfen wurde. Dieses Standardprotokoll stellte sich als Erfolg heraus, aus diesem wurde durch mehrere Änderungen und Funktionsausweitungen schließlich das NMEA 0183 Protokoll entwickelt.
Dieses Protokoll bildet die Grundlage der Kommunikation zwischen einem GPS-Empfänger und einem Rechner. Wenn im folgenden von NMEA-Protokoll gesprochen wird, ist das NMEA 0183 Protokoll gemeint. In diesem werden die Informationen, die vom oder an den Empfänger gesendet werden in spezielle NMEA-Rahmen gepackt. Das Format dieser Rahmen läßt sich folgendermaßen beschreiben:

Der Aufbau jedes Rahmens besteht aus einem ‚$'-Zeichen, gefolgt von einer zweistelligen „Talker-ID" und einer dreistelligen Rahmen-ID. Daran schließen sich die eigentlich zu übermittelnden Daten an, die jeweils in bestimmten, durch Kommata voneinander getrennten Feldern des Rahmens transportiert werden. Abgeschlossen wird ein Rahmen von einer optionalen Prüfsumme und einem obligatorischen Carriage Return/Line Feed. Die Maximallänge eines Rahmens beträgt 82 Zeichen, inklusive dem ‚$' und dem CR/LF. Dabei werden folgende Konventionen beachtet:

- Sollten die Daten, mit denen ein bestimmtes Feld in einem bestimmten Rahmen gefüllt werden soll nicht zur Verfügung stehen, dann wird dieses Feld trotzdem gesendet. Dies ist dann an einem leeren Feld, also zwei unmittelbar aufeinander folgenden Kommata, zu erkennen. Dies hat bei der Verwendung des NMEA-Protokoll die Konsequenz, daß der Zugriff auf bestimmte Felder durch Abzählen der Kommata geschieht, da jedes Feld in einem bestimmten Rahmen immer die gleiche Anzahl an Kommata vor und nach sich hat.
- Die optionale Prüfsumme besteht aus einem ‚*'-Zeichen, gefolgt von zwei Hexadezimalwerten, die durch eine XOR-Operation auf alle Zeichen zwischen dem ‚$' und dem ‚*' (exklusive der beiden Zeichen) berechnet werden. Die Prüfsumme ist bei manchen Rahmen obligatorisch.
- Der Standard erlaubt es Herstellern von GPS-Empfängern, eigene Rahmen zu definieren und einzusetzen. Ein so definierter Rahmen wird „proprietär" genannt. Man erkennt proprietäre Rahmen an einem unmittelbar auf das ‚$' folgendem ‚P'. Daran schließt sich eine dreistellige Hersteller-ID an, daran wiederum die vom Hersteller definierten Daten. Selbstverständlich muß die Darstellung dieser Daten dem NMEA-Standard genügen, z.B. müssen die einzelnen Felder durch Kommata voneinander getrennt werden.
  Ein Beispiel für eine Hersteller-ID ist GRM, das für Garmin steht. Jeder proprietäre Rahmen der Firma Garmin beginnt also mit „$PGRM", woran sich dann noch eine zusätzliche Kennung anschließt, die die endgültige Bedeutung des Rahmen festlegt, z.B. ‚E'.

Anhand des in NMEA 0183, Version 2.0 definierten Rahmens „$GPRMC" soll nun noch der Aufbau und exemplarisch die Bedeutung der Datenfelder der Rahmen verdeutlicht werden. Die Rahmenkennung „RMC" steht für *Recommended Minimum GPS/Transit Data*, dieser Rahmen wurde demnach als Träger einer empfohlenen Mindestinformation entworfen, die für die Navigation als notwendig erachtet wurde.

Der Rahmen:
$GPRMC,143029,A,4844.6196,N,00905.8400,E,000.0,000.0,230500,000.2,W*7C

Dies läßt sich aus ihm herauslesen:

- Zeitpunkt, zu dem der Rahmen erzeugt wurde: 14:30:29 GMT.
- Die aktuell ermittelte Position war gültig.
- Die geografische Breite der aktuellen Position war 48° 44,6196' Nord.
- Die geografische Länge der aktuellen Position war 9° 5,8400' Ost.
- Der Empfänger befand sich in Ruhe und hatte dementsprechend auch keinen eingeschlagenen Kurs.
- Dieser Rahmen wurde am 23.5.2000 verwendet.
- Die örtlich bestehende Variation betrug 0.2 in Richtung Westen.

Weitere Informationen, die sich aus NMEA 0183 Rahmen bestimmen lassen, sind u.a.:

- Höhenangaben.
- Informationen über die Art der durchgeführten Positionsbestimmung (GPS oder DGPS).

- Informationen über die Sendestation, die die letzte Korrekturnachricht sendete.
- Informationen, die die Qualität der durchgeführten Positionsbestimmung betreffen. Dies ist z.B. die Angabe der Metriken PDOP, HDOP, VDOP und TDOP, oder der Dimension der Positionsbestimmung (zwei- oder dreidimensional).
- Informationen, die die Satelliten, mit denen die Position bestimmt wurde betreffen, z.B. die Angabe der Anzahl der „sichtbaren“ Satelliten, der Anzahl der Satelliten, die tatsächlich verwendet wurden, sowie deren ID-Nummern.
- Wegpunktdaten, wie z.B. die Angabe des nächsten auf der eingegebenen Strecke liegenden Wegpunkts, ob dieser bereits erreicht wurde, oder in welcher Richtung dieser liegt.

Informationen, die durch Verwendung proprietärer Rahmen zusätzlich verfügbar sind, wären z.B. (hier nur Garmin-propritäre Rahmen):

- Angabe des berechneten Positionierungsfehler in horizontaler und vertikaler Richtung, sowie des daraus resultierenden Gesamtfehlers.
- Aufschlüsselung der Gesamtgeschwindigkeit in ihre beiden Komponenten (horizontal und vertikal)

Weiterhin wird mittels proprietärer Rahmen die Steuerung der Empfänger eines jeweiligen Herstellers durchgeführt, z.B. die Einstellung der zu verwendenden Baudrate für die Datenübertragung über eine serielle Schnittstelle, oder des den Positionsangaben zugrunde liegenden geodätischen Datums am Empfänger.
Genauere Angaben über die verschiedenen NMEA-Rahmen und die Daten, die diese jeweils enthalten, sind in Kapitel 5.1 tabellarisch dargestellt.

3 Grundlagen der Positionsbestimmung

In diesem Kapitel sollen die Grundzüge der Positionsbestimmung verständlich gemacht werden. Dazu werden als erstes die bei der Angabe von Positionen zugrundeliegenden Positionierungsmodelle erläutert. Anschließend werden die für die Bestimmung der Position mittels GPS wichtigen Begriffe und Modelle erklärt und jeweils mit einem Beispiel illustriert.

3.1 Positionierungsmodelle

Modelle zur Positionierung bilden die Basis jeder Positionsbestimmung. Ihre Funktion bei der Positionierung ist vergleichbar mit den Zahlensystemen der Mathematik, ohne die keine Berechnungen möglich wären. Es existieren eine Vielzahl von Positionierungsmodellen, die sich allerdings in 2 Klassen aufspalten lassen, die der geometrischen Modelle und die der symbolischen Modelle, die auf einer eindeutigen Benennung von diskreten, zellenbasierten Positionen mit Hilfe meist menschenlesbarer Bezeichner beruhen.
Die geometrischen Modelle lassen sich weiter in 3 Unterklassen aufteilen:

- Earth-Centered, Earth-Fixed XYZ-Koordinatensysteme, die eine Position mittels der Angabe einer Koordinate in einem (dreidimensionalen) kartesischen Koordinatensystem festlegen.
- Geografische Koordinatensysteme, die eine Position durch Angabe dreier Werte, der geografischen Breite, der geografischen Länge und einer Höhe festlegen.
- Konforme Koordinatensysteme, die die Erdoberfläche in (zweidimensionale) Teilflächen aufteilen, und die Position durch Angabe von sogenannten Rechts- bzw. Hochwerten bezogen auf eine dieser Teilflächen angeben.

In dieser Arbeit wird im Detail nur auf die geometrischen Modelle eingegangen, und hier im speziellen auf die geografischen Koordinatensysteme, die auch vom GPS-Empfänger und damit auch in der Schnittstelle verwendet werden. Die anderen geometrischen Modelle werden der Vollständigkeit halber dennoch erläutert, außerdem wird kurz die Möglichkeit der Konvertierung zwischen manchen dieser Modelle umrissen.
Zuerst werden allerdings die allen geometrischen Modellen zugrundeliegenden geodätischen Daten näher erläutert, da das Verständnis dieser die Voraussetzung zum Verständnis jedes geometrischen Modells bildet.

3.1.1 Geodätische Daten

Alle geometrischen Modelle haben gemeinsam, daß sie die Position mit Hilfe einer Koordinate in einem Koordinatensystem beschreiben. Diese Koordinatensysteme sind als Referenzkoordinatensysteme zu verstehen, die auf einer Beschreibung der Erde als abgeflachte Kugel beruhen. Man spricht hier von Referenzellipsoiden. Die Ellipsoidenform der Erde ist wiederum durch die Erdrotation bedingt, genauer durch die Fliehkräfte, die von dieser erzeugt werden. Diese haben zur Folge, daß die eigentlich kugelförmige Erde an den Polen um ungefähr 20 Kilometer abgeflacht ist. Ein Ellipsoid wird eindeutig festgelegt durch die Angabe zweier Werte, der Länge der *Haupthalbachse a* und die Länge der *Nebenhalbachse b*. Oft werden auch die Länge der Haupthalbachse und die *Abflachung* des Ellipsoiden *f*, die sich zu $f = (a-b)/a$ berechnet, angegeben. Die Länge der Haupthalbachse entspricht bei der Erde dem Äquatorradius, die Länge der Nebenhalbachse dem Polradius. Dabei ist zu beachten, daß die Genauigkeit der Positionsbestimmung abhängt von der

Genauigkeit der Referenzkoordinatensysteme, welche wiederum vom zugrunde gelegten Erdmodell abhängen. Man kann also feststellen, daß je genauer das durch einen Ellipsoiden der Gestalt der Erde angenäherte Erdmodell der tatsächlichen Form der Erde gleicht, sich die Genauigkeit der Positionsbestimmung dem theoretisch möglichen Optimum angleicht. In der untenstehenden Abbildung werden diese Parameter verdeutlicht. Referenzellipsoiden neueren Datums gleichen der tatsächlichen Form der Erde bis auf 100 Meter genau. Als Beispiel werden hier die Parameterwerte für den WGS84-Ellipsoiden, der dem WGS84-Datum (WGS = **W**orld **G**eographic **D**atum von 1984) zugrunde liegt, angegeben. Das WGS84-Datum gehört zur Klasse der geografischen Koordinatensysteme.

Parameterwerte des WGS84-Ellipsoiden:

Länge der Haupthalbachse a = 6356752,3142 Meter
Länge der Nebenhalbachse b = 6378137,0 Meter
Abflachung $f = (a-b)/a$ = 1/298,257223563

Die Parameterwerte für die anderen, den existierenden geografischen Koordinatensystemen zugrunde liegenden Referenzellipsoiden, sind unter anderem bei [DAN98] zu finden.

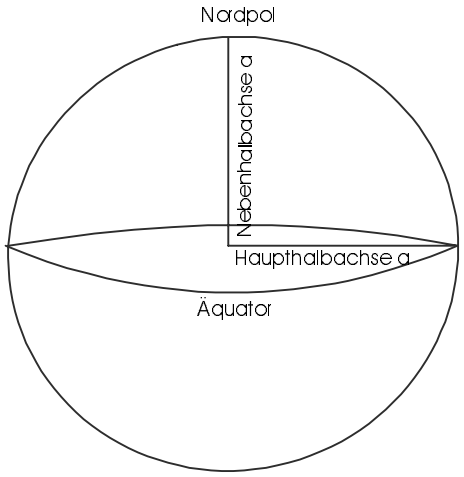


Abbildung 9: Die Erde als Ellipsoid repräsentiert

Im Zusammenhang mit den geodätischen Daten fällt oft auch der Begriff *Geoid*. Bei einem Geoid handelt es sich um ein Modell der Erdoberfläche, das versucht, die in Wirklichkeit unregelmäßige Oberfläche der Erde auf eine Fläche abzubilden, die allein durch die Auswirkungen der Erdgravitation entstehen würde. Diese ist ebenfalls nicht eben, denn es existieren Unterschiede im Erdkern und in der Oberflächenstruktur der Erde, die zu unterschiedlichen Gravitationsvektoren führen. Ein Geoid ist also ein theoretischer Körper, mit dessen Hilfe man den tatsächlichen Oberflächenverlauf modellieren kann.

3.1.2 Earth-Centered, Earth-Fixed XYZ-Koordinatensyteme

Diesen Koordinatensystemen liegt ein dreidimensionales kartesisches Koordinatensystem zugrunde. Dieses Koordinatensystem hat seinen Ursprung im Massezentrum des jeweiligen Referenzellipsoids. Die 3 Achsen werden dann folgendermaßen definiert:

- Die Z-Achse verläuft durch Ursprung und Nordpol.

- Die X-Achse wird durch die Schnittgerade der durch den Hauptmeridian festgelegten und der Äquatorebene festgelegt.
- Die Y-Achse verläuft rechtwinklig und östlich zur Z-Achse und schneidet den Äquator.

Nach dieser Definition eines Koordinatensystems ergibt sich die Koordinate eines Punktes als die Komponenten des Richtungsvektors zwischen dem Ursprung und dem Punkt. Als Einheit wurden Meter gewählt.

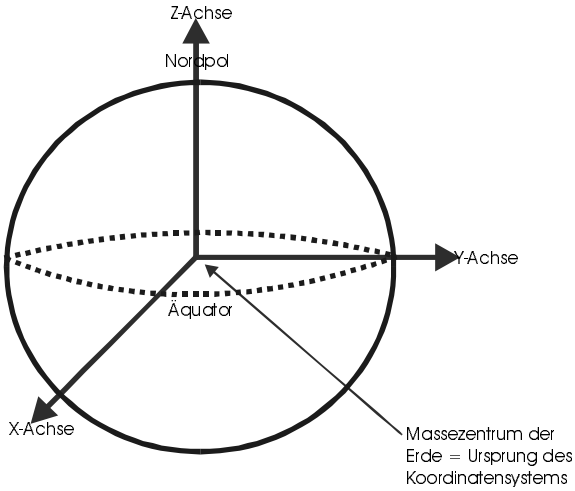


Abbildung 10: Earth Centered, Earth Fixed Koordinatensystem

3.1.3 Geografische Koordinatensysteme

Diese Unterklasse der geometrischen Modelle ist die wohl am weitesten verbreitete Form der Positionsangabe. Eine Position wird eindeutig bestimmt durch die Angabe einer geografischen Breite, einer geografischen Länge und einer geografischen Höhe, die orthogonal zur Oberfläche des Ellipsoiden gemessen wird. Für die Festlegung dieser Größen wurden Referenzebenen festgelegt, auf die sich Längen- und Breitenangaben beziehen. Als Referenzebene für die geografische Breite wurde die Ebene gewählt, die den durch Greenwich (GB) verlaufenden Hauptmeridian vollständig enthält, für die geografische Länge die Ebene, die den Äquator vollständig enthält. Dann definiert sich die geografische Breite eines Punktes durch den Winkel von der Äquatorebene zur vertikalen Richtung einer Linie, die waagerecht zur Oberfläche des Referenzellipsoiden ist und die durch den Punkt geht. Die geografische Breite ist definiert durch den Winkel zwischen der Hauptmeridianebene und einer anderen Ebene, die den Punkt enthält und ebenfalls senkrecht zur Äquatorebene steht. Für die Angabe einer Position in einem geografischen Koordinatensystem existieren mehrere gängige Formate. Häufig wird die Darstellung in ° (Grad), ´ (Winkelminuten) und ´´ (Winkelsekunden) für Länge und Breite gewählt, evtl. noch mit Angabe der Hemisphäre (Nord bzw. Süd für die Breite und Ost bzw. West für die Länge), wenn dies nicht schon aus dem Vorzeichen der Gradangabe ersichtlich ist. Es existieren aber noch andere gebräuchliche Formate, z.B. das „dd.dddd“ Format, in dem eine Länge oder Breite durch eine Dezimalzahl angegeben wird, deren ganzzahliger Anteil als Anzahl der vollen Grade und deren Dezimalteil als Bruchteil eines vollen Grads gesehen wird. Nicht unüblich ist auch die Darstellung gemäß einem „dd.mmmm“-Formats, in dem der Dezimalteil in Winkelminuten angegeben wird. Die beiden letzten Formate werden häufig eingesetzt, wenn mit den Daten Berechnungen durchgeführt werden müssen, z.B. die Bestimmung des Abstands zweier Positionen. Abbildung 11 illustriert die geografischen Koordinatensysteme nochmals.

Geografische Koordinatensysteme werden z.B. als Standard in den meisten gebräuchlichen GPS-Empfängern verwendet, unter diesen hat sich das sich auf den WGS84-Ellipsoiden beziehende geodätische Datum WGS84 durchgesetzt. Bei Verwendung dieses geodätischen Datums als Arbeitsdatum erhält der Benutzer den Vorteil, daß keine Konvertierung mehr von den vom Empfänger ausgegebenen Positionsdaten auf das vom Benutzer verwendete geodätische Datum durchgeführt werden muß.

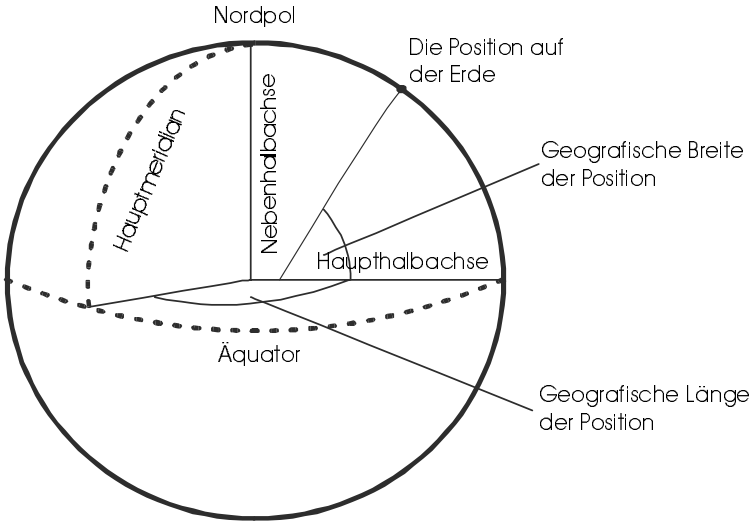


Abbildung 11: Definition geografischer Breite und Länge

3.1.4 Konforme Koordinatensysteme

Die konformen Koordinatensysteme basieren auf den geografischen Koordinatensystemen, genauer gesprochen stellen sie lediglich eine alternative Darstellungsweise der letzteren dar. Der Vorteil der konformen Koordinatensysteme gegenüber den herkömmlichen geografischen Systemen besteht darin, daß innerhalb gewisser Grenzen Positionsberechnungen wie in einem bekannten rechtwinkligen Koordinatensystem erfolgen können. Man versucht dazu, ein metrisches rechtwinkliges Koordinatensystem auf die gekrümmte Oberfläche des gewählten Referenzellipsoiden zu legen. Es ist leicht einzusehen, daß dieses System nur korrekt sein kann, wenn der Ausschnitt der durch ein Element des durch das Koordinatensystem definierten Gitternetzes überdeckten Ellipsoidenoberfläche hinreichend klein ist. Deshalb haben alle bekannten konformen Koordinatensysteme eine maximale horizontale Ausdehnung ihrer Gitterelemente von ca. 3 bis 6 Grad. Könnte man ein konformes Koordinatensystem von der Erde „abziehen“ und würde es flach ausrollen, würde sich ein Bild ähnlich Abbildung 12 ergeben.

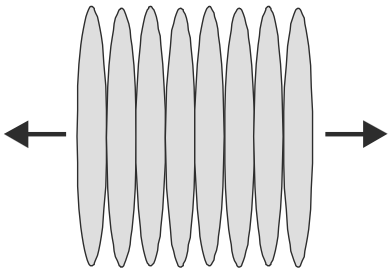


Abbildung 12: Gitternetz eines konformen Koordinatensystems

Jede Positionsangabe bezieht sich nun auf eines dieser Elemente. Die einzelnen Elemente werden dabei jeweils symmetrisch zu den sogenannten Mittelmeridianen plaziert, sie werden deshalb auch als *Meridianstreifen* bezeichnet. Die Ausdehnung der Meridianstreifen des in Deutschland gebräuchlichsten konformen Koordinatensystems, des *Gauß-Krüger*-Koordinatensystems beträgt 3° pro Streifen, dies resultiert in 120 verschiedenen Meridianstreifen. Beim international eingesetzten *Universal Transverse Mercator-System (*kurz: *UTM*) existieren dagegen 60 Meridianstreifen (Ausdehnung von 6° pro Streifen), die zur Unterscheidung von 0 bis 59 durchnumeriert sind. Aus Übersichtsgründen wird ein einzelner Streifen noch horizontal in Zonen geteilt, beim UTM-System z.B. in zumeist 8° große Teile. Dadurch wird eine Reihe von Zonen definiert, die die Angabe einer Position vereinfachen. Um eine Position in einem konformen Koordinatensystem eindeutig angeben zu können ist dann folgendes nötig:

- Angabe der Zonennummer
- Angabe eines Rechts- und eines Hochwerts

Der Rechts- oder Ostwert (E) ist dabei der senkrechte Abstand vom Mittelmeridian des Meridianstreifens, in dem sich die betrachtete Zone befindet. Der Hoch- oder Nordwert (N) ist definiert durch den Abstand zum Äquator, am Mittelmeridian entlang gemessen. Zur Vermeidung negativer Werte für den Rechtswert hat sich die Konvention durchgesetzt, den Rechtswert jedes Mittelmeridians auf den Wert 500.000 Meter zu setzen. Negative Rechtswerte, also Positionen die links vom Mittelmeridian liegen, haben demnach einen Wert kleiner als 500.000 aber größer null, positive Rechtswerte einen Betrag größer als 500.000. Der Rechtswert des Punktes in untenstehender Abbildung hätte also den Wert der entsteht, wenn von 500.000 der Betrag des Abstandes vom Mittelmeridian abgezogen wird. Diese Konvention wird häufig als „*false Easting*“ bezeichnet, da der angegebene Ostwert nicht mit dem tatsächlichen übereinstimmt, sondern jeweils um 500.000 Meter verschoben ist.

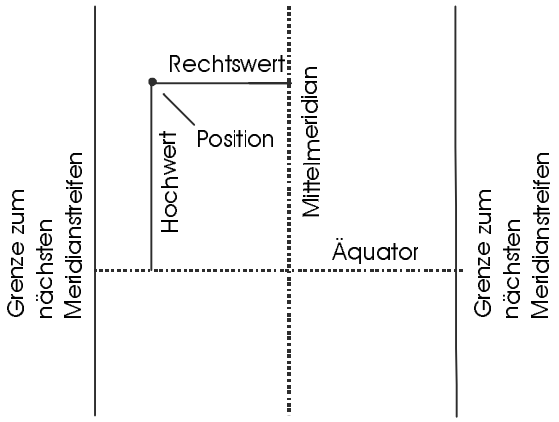


Abbildung 13: Definition des Rechts- und Hochwertes

3.1.5 Konvertierungen zwischen geografischen Modellen

Es besteht die Möglichkeit, zwischen manchen dieser verschiedenen Systeme zu konvertieren. Dabei soll hier nur die Konvertierung zwischen Koordinatendarstellungen betrachtet werden, die auf demselben Referenzellipsoiden basieren. Im anderen Fall, in dem die Systeme voneinander verschiedene Referenzellipsoiden benutzen, läßt sich eine Konvertierung nur dann durchführen, wenn gewisse Transformationsbeziehungen zwischen diesen Ellipsoiden bekannt sind. Diese Beziehungen bestehen z.B. aus einer Menge von Punkten, deren

Koordinaten bereits in beiden Koordinatensystemen bekannt sind und einem mathematischen Modell, das die Transformation formal beschreibt. Ist dies erfüllt, dann läßt sich auch hier eine Koordinatentransformation durchführen. In diesem Zusammenhang sind dann die sogenannten Standard-Molodensky-Formeln ein Begriff. Für eine weiterführende Betrachtung dieser Art der Koordinatentransformation sei hier z.B. auf [DAN98] verwiesen.
Bei Verzicht auf die erforderliche Transformation, und Interpretation von Koordinaten, die sich auf einen bestimmten Referenzellipsoiden beziehen, als Koordinaten in einem sich auf einen anderen Referenzellipsoiden beziehenden Koordinatensystem, würde sich ein relativer dreidimensionaler Fehler von bis zu einen Kilometer ergeben, was natürlich für die meisten Anwendungen inakzeptabel ist.
Konvertierungen zwischen Koordinatendarstellungen mit demselben Referenzellipsoiden sind einfacher durchzuführen, allerdings existiert eine Konvertierungsreihenfolge, die eingehalten werden muß. So sind Konvertierungen nur in der Reihenfolge *Earth Centered/Earth Fixed KS, geografische KS, konforme KS* möglich, wobei hier beide Richtungen möglich sind.

- **EC/EF ⇔ geografische KS**:
  Die hierfür nötige Transformation läßt sich zurückführen auf die bereits in der Schule gelehrte Umrechnung von kartesischen Koordinaten in polare Koordinaten oder umgekehrt. Dabei wird der Massepunkt der Erde sowohl mit dem Ursprung des die EC/EF Koordinaten beschreibenden kartesischen Koordinatensystems, als auch mit dem Mittelpunkt der Kugel, mit deren Hilfe Polarkoordinaten im dreidimensionalen Raum beschrieben werden, identifiziert.
  Punkte eines geografischen KS lassen sich dann gemäß der folgenden Formeln in Punkte eines EC/EF KS transformieren:

  $$X = r * cos(\phi) * cos(\theta);\ Y = r * cos(\phi) * sin(\theta);\ Z = sin(\phi).$$

  Dabei bezeichnet der Winkel ϕ die geografische Länge, der Winkel θ die geografische Breite und r den Abstand des Punktes vom Massepunkt der Erde.
  Formeln für die andere Richtung der Transformation werden u.a. bei [SNY93] angegeben.

- **Geografische KS ⇔konforme KS**
  Diese Transformation kann durch eine vorhergehende Approximation durch eine Potenzreihe angegangen werden. Mit Hilfe dieser Potenzreihe wird versucht, die für die Transformation nötigen Differentialgleichungen möglichst genau anzunähern. Die eigentliche Transformation findet dann durch Lösung dieser Potenzreihe statt. Näheres zu diesen Transformationen ist ebenfalls z.B. in [SNY93] zu finden.

3.2 Grundlagen der Positionsbestimmung und Navigation

In diesem Unterkapitel wird dem Leser das benötigte Grundwissen näher gebracht, das er braucht, um die Information die manche NMEA-Rahmen enthalten, zu verstehen und selber zu verwenden. Im wesentlichen handelt es sich dabei um eine Erklärung der Begriffe *True Course (*dt.: Wahrkurs)*, Magnetic Course (*dt.: Magnetischer Kurs) und der Vollständigkeit halber auch *Compass Course (*dt.: Kompaßkurs), obwohl dieser in der Schnittstelle keine Verwendung findet. Diese stellen die unterschiedlichen Möglichkeiten dar, einen verfolgten Kurs aufzufassen. Anschließend wird noch kurz auf die bereits früher angesprochenen, unter

dem Begriff *GDOP (Geometric Dilution of Precision,* dt.: Geometrische Positionspräzisionsminderung) zusammengefaßten Begriffe *HDOP, VDOP* und *PDOP* (dt.: Horizontale-, Vertikale-, Positionspräzisionsminderung) eingegangen, die Metriken zur Einordnung der Genauigkeit einer Positionsangabe darstellen. An dieser Stelle soll noch angemerkt werden, daß hier und im folgenden, wenn für nötig befunden, statt der entsprechenden deutschen die (originalen) englischen Begriffe verwendet werden. Dies soll einem Leser und späteren Benutzer der entstandenen Kommunikationsbibliothek die Wiedererkennung der in der Schnittstelle verwendeten englischen Begriffe erleichtern.

3.2.1 True course, magnetic course & compass course

Zur Abgrenzung dieser drei Begriffe untereinander werden noch einige andere Begriffe benötigt, dies sind:

- *Abweichung (*engl. *Deviation)*
  Die *Abweichung*, die jeder Kompaß besitzt, hängt ab von den ihn umgebenden Umständen, sie kann z.B. durch elektrische Geräte oder Stromkabel beeinflußt werden. Die Richtung der *Abweichung* kann entweder nach Osten oder nach Westen sein, dies hängt vom jeweiligen Kurs ab, den der Kompaß(-träger) innehat, die *Abweichung* kann außerdem an verschiedenen Positionen unterschiedlich groß sein. Man kann nun Abweichungstabellen erstellen, an der die an einem Punkt vorherrschende *Abweichung* eingetragen und später nachgeschlagen werden kann. Die *Abweichung* ist ein Maß für den Unterschied zwischen der vom Kompaß angezeigten Nordrichtung und der tatsächlichen Richtung, in der der magnetische Nordpol liegt.
- *Variation*
  Die Variation ist ein Maß für den Unterschied zwischen den Werten für den *magnetischen Norden* und den *geografischen Norden*. Der Begriff *magnetischer Norden* wiederum definiert sich zu der Richtung, in die ein Kompaß zeigen würde, wenn es an diesem Ort keine *Abweichung* gäbe. Der Begriff *geografischer Norden* definiert sich als die Richtung in welcher der geografische Nordpol liegt. Dieser Unterschied verändert sich Jahr für Jahr geringfügig und ist ortsabhängig. Der Unterschied kann entweder eine Abweichung nach Osten oder eine Abweichung nach Westen sein. Bootsfahrer sollten mit diesem Begriff vertraut sein, zeigt doch jeder Bootskompaß den Wert der *Variation* in Grad an.

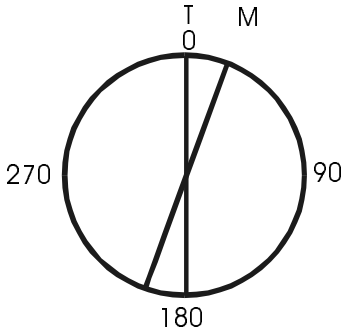


Abbildung 14: die Variation auf einem Bootskompaß

Die obige Abbildung erläutert dies noch einmal, wobei mit T die Richtung des geografischen Nordpols und mit M die Richtung des magnetischen Nordpols bezeichnet ist. An ihr ist zu erkennen, daß die *Variation* größer wird, je weiter man sich dem Nordpol nähert und kleiner wird, je weiter man sich dem Äquator nähert.

Nachdem diese Begriffe geklärt wurden, folgt nun die Erklärung der einzelnen Kursarten:

- *Compass Course*
  Dies ist die Kursart, die sich von einem handelsüblichen Kompaß ablesen läßt. Sie ist Grundlage für die Definition der beiden anderen Kursarten.
- *Magnetic Course*
  Ergibt sich aus dem *Compass Course* durch Addition der örtlich vorherrschenden und im Vorzeichen durch die aktuelle Fahrtrichtung bestimmte *Abweichung*.
- *True Course*
  Wird aus einem *Magnetic Course* durch Addition der örtlich und zeitlich vorherrschenden *Variation* bestimmt.

Das Verfahren, das der Umrechnung zwischen den unterschiedlichen Kursarten jeweils zugrunde liegt, läßt sich durch die nachfolgend in Pseudocode angegebenen Algorithmen beschreiben.

1. Umrechnung eines *True Course* in einen *Compass Course*. Diese Umrechnung wird immer dann nötig sein, wenn aus einem z.B. mit einem GPS-Empfänger bestimmten *True Course* der tatsächlich physisch einzuschlagende Kurs bestimmt werden soll. Als Beispiel kann hier ein Wanderer gelten, der die Richtung zum nächsten Wegpunkt mit einem mitgeführten GPS-Empfänger ermittelt hat und der nun den Kurs berechnen will, dem er folgen muß, um diesen Wegpunkt zu erreichen. Die benötigten Werte für die *Abweichung* und die *Variation* sind im ersten Fall z.B. über im GPS-Gerät integrierten Datenbanken erhältlich, im zweiten Fall z.B. direkt aus den vom GPS-Empfänger zur Verfügung gestellten Navigationsdaten.

   ```
   PROCEDURE TrueToCompass (VAR TrueCourse, VAR Variation, VAR Abweichung)

       VAR MagneticCourse;
       VAR CompassCourse;

   BEGIN (*PROC*)

       IF (Richtung der Variation = ‚West‘)
               MagneticCourse := TrueCourse + Variation;
       ELSE
               MagneticCourse := TrueCourse – Variation;

       IF (Richtung der Abweichung = ‚West‘)
               CompassCourse = MagneticCourse + Abweichung;
       ELSE
               CompassCourse = MagneticCourse – Abweichung;

       RETURN CompassCourse;

   END; (*PROC*)
   ```

2. Umrechnung eines *Compass Course* in einen *True Course*. Dies ist immer dann nötig, wenn nicht der physisch eingeschlagene Kurs, sondern eine von den Einflüssen von *Abweichung* und *Variation* bereinigte Kursart von Interesse ist. Dies ist z.B. der Fall,

wenn ein Kursverlauf in eine grafische Karte eingezeichnet werden soll, beispielsweise um eine zu vermessende Strecke darzustellen. Dabei ist der *Compass Course* durch Blick auf einen mitgeführten Kompaß ersichtlich, die Werte für die *Abweichung* bzw. die *Variation* können z.B. bei der Auswertung der gesammelten Daten aus einer Datenbank entnommen werden bzw. eventuell ebenfalls vom Kompaß abgelesen werden.

```
PROCEDURE CompassToTrue(VAR CompassCourse, VAR Variation, VAR Abweichung)

    VAR MagneticCourse;
    VAR TrueCourse;

BEGIN (*PROC*)

    IF (Richtung der Abweichung = ‚West‘)
            MagneticCourse := TrueCourse - Abweichung;
    ELSE
            MagneticCourse := TrueCourse + Abweichung;

    IF (Richtung der Variation = ‚West‘)
            TrueCourse = MagneticCourse - Variation;
    ELSE
            TrueCourse = MagneticCourse + Variation;

    RETURN TrueCourse;

END; (*PROC*)
```

Im folgenden sollen die oben angegebenen Algorithmen mit zwei kleinen Beispielen veranschaulicht werden.

- **Berechnung des *Compass Course* aus einem bekannten *True Course*:**

  Bekannte Daten: *True Course* = 120°, *Abweichung* = 10° West, *Variation* = 5° Ost
  Berechnungsfolge: → *Magnetic Course* = 120° - 5° = 115°
  → *Compass Course* = 115° + 10° = **125**°

- **Berechnung des *True Course* aus einem bekannten *Compass Course:***

  Bekannte Daten: *Compass Course = 125°, Abweichung* =10° West*, Variation* = 5° Ost
  Berechnungsfolge: → *Magnetic Course* = 125° - 10° = 115°
  → *True Course* = 115° + 5° = **120**°

3.2.2 Die Präzisionsminderung (*DOP)

Eine wichtige Metrik, um Positionsangaben, die von einem GPS-Empfänger bestimmt wurden, bezüglich ihrer Genauigkeit einordnen zu können, ist die in verschiedenen Ausprägungen von jedem GPS-Empfänger zur Verfügung gestellte sogenannte

*Präzisionsminderung (*engl. ***D**ilution **of** **P**recision*). Im folgenden wird diese abkürzend als *DOP* bezeichnet. Die Bedeutung der DOP wurde bereits in 2.3.4 unter dem Begriff *Geometric Dilution of Precision (*kurz: *GDOP*) ansatzweise erläutert. Die *GDOP* ist demnach eine Metrik für die Unschärfe, die bei jeder Positionsbestimmung in unterschiedlichem Maße auftritt. Mathematisch gesprochen, ist die *GDOP* definiert als Quadratwurzel der Summe der Varianzen der Position- und Zeitfehlerabschätzungen. Dies läßt sich durch folgende Formel ausdrücken:

$$GDOP = (s_x^2 + s_y^2 + s_z^2 + c^2 s_t^2)^{0,5},$$

wobei hier mit ‚c' die Lichtgeschwindigkeit und mit ‚t' die Uhrendrift der Empfängeruhr bezeichnet wird. Zusätzlich zur allgemeinen *GDOP* wurden noch andere, für die Positionsbestimmung wichtige Metriken definiert. Diese sind unter anderem *PDOP* (**P**osition DOP), *HDOP* (**H**orizontal DOP), *VDOP* (**V**ertical DOP) und *TDOP* (**T**ime DOP). Die Bedeutung und die Formeln, aus denen sich die entsprechenden Werte bestimmen lassen, können werden im folgenden angegeben:

- *PDOP* ist definiert als Quadratwurzel der Varianzen der Positionsfehlerabschätzungen.

$$PDOP = (s_x^2 + s_y^2 + s_z^2)^{0,5}$$

- *HDOP* ist definiert als Quadratwurzel der Varianzen der horizontalen Positionsfehlerabschätzungen.

$$HDOP = (s_x^2 + s_y^2)^{0,5}$$

- *VDOP* ist definiert als Quadratwurzel der Varianz der vertikalen Positionsfehlerabschätzung.

$$VDOP = (s_z^2)^{0,5}$$

- *TDOP* ist definiert als Quadratwurzel der Varianz der Zeitfehlerabschätzung.

$$TDOP = (s_t^2)^{0,5}$$

Durch die Definition der oben angegebenen Metriken wurde den Benutzern des GPS-Systems die Möglichkeit gegeben, bestimmte Positions- und Zeitangaben auf ihre Genauigkeit hin zu prüfen und einzuordnen.

4 Entwurf einer NMEA 0183 konformen Schnittstelle

Hauptaufgabe dieser Studienarbeit war der Entwurf und die Realisierung einer Schnittstelle, die es ermöglichen sollte, Daten von einem den NMEA 0183 Standard verwendenden GPS-Empfänger zu erhalten. Es wurden dabei einige Vorgaben in Bezug auf das Betriebssystem (Microsoft Windows 95/98), den Zielrechner (Xybernaut Mobile Assistant), die Programmiersprache (Java 2) usw. gestellt, die allerdings erst im nächsten Kapitel, das die eigentliche Realisierung der Aufgabe beschreibt, im Detail besprochen werden. In diesem Kapitel werden vorerst nur die allgemeingültigen Entwurfsentscheidungen und ihre Gründe erläutert werden.

4.1 Den Entwurf beeinflussende Randbedingungen

4.1.1 Allgemeine Vorgaben

Diese Vorgaben wurden zu Anfang der Studienarbeit gestellt, wobei jeweils eine Überprüfung auf die Realisierbarkeit einer einzelnen Vorgabe vorgenommen werden sollte.

- Die Schnittstelle sollte möglichst mit jedem NMEA 0183 konformen Empfänger auch ohne vorhergehende Erweiterung zu verwenden sein. Dazu sollte wenn möglich ganz auf die Verwendung proprietärer Rahmen verzichtet werden.
- Da für eine zuverlässige Positionsbestimmung eine hohe Genauigkeit, oder zumindest das Wissen über die Qualität der Positionsbestimmung unabdingbar ist, wurde insbesondere die Einführung eines jederzeit abfragbaren Status gefordert, anhand dessen der aktuelle Systemzustand des Empfängers erkennbar ist.
- Zusätzlich zur eigentlichen Schnittstelle sollte ein Utility entworfen und implementiert werden, mit deren Hilfe man ein GPS-Gerät und dessen Eigenschaften testen kann. Diese sollte es unter anderem ermöglichen, sich gesammelte Navigationsdaten wie z.B. die Position, Geschwindigkeit und den Abstand zu einer anderen, selbst bestimmbaren Position, anzeigen zu lassen. Die Möglichkeit der Erzeugung einer Logdatei, mit deren Hilfe sich zurückgelegte Strecken rekonstruieren lassen, wurde ebenfalls verlangt. Weiterhin sollte noch ein Tool entworfen werden, mit dessen Hilfe die aktuelle Position in eine grafische Karte eingezeichnet werden kann. Die Beschreibung der Umsetzung dieser Vorgabe und der Funktionalität der beiden Anwendungen wird in einem eigenen Kapitel vorgenommen.
- Als Programmiersprache war Java 2 zu verwenden.

4.1.2 Besonderheiten des NMEA 0183 Protokolls und ihre Konsequenzen

Das NMEA 0183 Protokoll bietet Herstellern von GPS-Geräten die Möglichkeit, eigene Datenrahmen zu spezifizieren. Ein Vorteil, die diese Möglichkeit mit sich bringt, ist die verbesserte Unterstützung, die die einzelnen Produkte der verschiedenen Herstellern erfahren. Da ein Hersteller selbst Datenrahmen definieren kann, ist es ihm möglich, die Besonderheiten und Verbesserungen seiner Geräte voll auszunutzen. Bietet das Gerät z.B. irgendeine neue Art von Information an, so kann ein neuer proprietärer Rahmen definiert werden, der eben diese Information in einem seiner Datenfelder transportiert. Ein Nachteil dieses liberalen Ansatzes ist durch das Vorgehen vieler Hersteller, Informationen, die eigentlich in allgemeinen Rahmen versendet werden, auch in selbst definierte proprietäre Rahmen einzufügen,

geschaffen worden. Die Folge ist eine große Menge Rahmen mit sich überschneidendem Informationsgehalt. Trotz der teils redundanten Informationen der proprietären Rahmen enthalten diese andere nützliche Daten, die nicht in allgemeinen Rahmen versendet werden. Die Überprüfung der weiter oben vorgestellten entsprechenden Vorgabe des Verzichts auf proprietäre Rahmen hatte damit ein negatives Ergebnis. Ein Verzicht auf proprietäre Rahmen würde unzulässig viel der vom Empfänger angebotenen Funktionalität verschenken. Um die Forderung der allgemeinen Verwendbarkeit dennoch zu erfüllen, wurde beim Entwurf der vorzustellenden Schnittstelle ein modulares, auf Vererbung basierendes, leicht erweiterbares Konzept gewählt, das sowohl den Erhalt allgemeiner NMEA-Rahmen, als auch einer gewissen Anzahl proprietärer Rahmen ermöglicht. Dieses Konzept erhält weitere Bestätigung aus dem Fehlen direkter Empfängersteuersätze im allgemeinen NMEA 0183 Protokoll. So wird z.B. kein an den Empfänger zu sendender Rahmen angeboten, der eine Auswahl unter den von diesem unterstützten Baudraten für den Datentransfer möglich macht. Die Realisierung dieser Konfigurationseinstellung wurde den Herstellern der einzelnen Empfänger überlassen. Um so mehr wird also die Notwendigkeit eines modularen Konzeptes mit einem allgemein gültigen und mehreren speziellen, an die verschiedenen Hersteller angepaßten, Modulen ersichtlich. Dieses jetzt noch recht abstrakte Konzept wird im nächsten Kapitel ausführlicher erläutert.
Da bei der Entwicklung dieser Schnittstelle aus Aufwandsgründen nicht die Auswertung aller verschiedener Rahmen implementiert werden konnte, wurde versucht, die am häufigsten von verschiedenen GPS-Geräten verwendeten Rahmen herauszufinden, wobei noch auf eine möglichst gute Überdeckung der unterschiedlichen Datentypen geachtet wurde. Die getroffene Auswahl berücksichtigt die meisten allgemeinen Rahmen nach NMEA 0183 V.2.0 und da ein Empfänger der Firma Garmin zum Test zur Verfügung stand, wurden auch noch die meisten von diesem Gerät unterstützten Garmin-proprietären Rahmen aufgenommen.

4.1.3 Allgemeine Randbedingungen

GPS-Empfänger versenden die von ihnen bestimmten Navigationsinformation gewöhnlich kontinuierlich, d.h. ohne merkbare Unterbrechung. Allerdings gibt es beim Senden der Rahmen verschiedene Auffälligkeiten:

1. Gewöhnlich ist der Strom von Rahmen in eine gewisse Reihenfolge geordnet, d.h. es gibt eine gewisse Sendereihenfolge von Rahmen, die allerdings bei jedem Empfänger anders sein kann. Es ist also nötig, einen eindeutig bestimmbaren Endpunkt eines Rahmenzyklus zu bestimmen, so daß man nach Erhalt dessen weiß, daß ein Zyklus zu Ende ist und die darauffolgend empfangenen Rahmen zu einem neuen Zyklus gehören. Dies ist deshalb wichtig, weil nur so ein bestimmter Rahmen, der eine bestimmte Information enthält, einer Gruppe von anderen Rahmen zugeordnet werden kann, deren Informationen vom Empfänger mit denselben Rohdaten berechnet wurden. Betrachtet man zusätzlich die Tatsache, daß Empfänger verschiedener Hersteller oftmals einen voneinander unterschiedlichen Rahmensatz besitzen und daß eine Grundanforderung an die zu entwerfende Schnittstelle die Verwendbarkeit mit Geräten verschiedener Hersteller war, dann wird augenscheinlich, daß zur Bestimmung eines solchen Endpunktes eines Zyklus ein Rahmen zu wählen ist, der von möglichst vielen verschiedenen Empfängern unterstützt wird. Deshalb war von vorne herein klar, daß ein solcher Rahmen ein allgemeiner NMEA 0183 konformer Rahmen sein muß, denn nur so ist gesichert, daß dieser Rahmen von den Empfängern möglichst vieler Hersteller verwendet werden würde. Anders gesagt, bestimmt die Wahl des Endrahmens die spätere Verwendbarkeit der Schnittstelle mit einem bestimmten Empfänger, denn wenn dieser den gewählten Rahmen nicht unterstützt, dann wird nie ein Zyklusende erkannt werden und damit keine

Information übernommen werden. Im nächsten Kapitel wird dann die konkrete und wichtige Wahl des „Endrahmens“ beschrieben werden.

2. In diesem Zusammenhang war es ebenfalls wichtig, in der Schnittstelle eine Möglichkeit der Selektion verschiedener Rahmen zu bieten. Oftmals ist für eine bestimmte Anwendung nicht die gesamte, vom Empfänger angebotene, Information von Belang, sondern nur eine Teilmenge dieser. Diese Wahl sollte sich direkt auf die vom Empfänger gesendeten Rahmen auswirken, denn das Senden nicht benötigter Information kostet Bandbreite und damit Zeit. Ein Benutzer soll also entscheiden können, ob er alle möglichen Daten vom Empfänger benötigt oder nicht, und der Empfänger soll diese Entscheidung beim Versenden neuer Rahmen berücksichtigen.
3. Obwohl es mit vielen Empfängern möglich ist, die Position auch gemäß anderen als den geografischen Koordinatensystemen darstellen zu können, ist es doch am weitesten verbreitet, die aktuelle Position als geografische Koordinate darzustellen. Da die Position empfängerintern meist als WGS84-Koordinate dargestellt wird, bietet sich die Verwendung eben dieses Formats auch extern an, d.h. bei Wahl eines geografischen Koordinatensystems zur Positionsangabe sind keine, oder nur geringe Transformationen nötig. In Kapitel 8 wird als mögliche Erweiterung dieser Arbeit die Implementierung neuer Modelle zur Positionierung vorgeschlagen.
4. Da ein NMEA 0183 konformer GPS-Empfänger besonders bei Freischaltung aller seiner angebotenen Rahmen eine große Menge an Informationen zurückliefert, wurde beschlossen, diese Informationen kompakt in einem eigenständigen Objekt abzuspeichern. Der Zugriff auf die einzelnen abgespeicherten Informationen ist dann indirekt über das Speicherobjekt und eine entsprechende Zugriffsmethode für die einzelne Information möglich.

Weitergehende und zum Teil auf die obigen Entscheidungen basierende Entwurfsentscheidungen waren:

- Da der Zugriff auf die gesammelten Daten eines Zyklus über ein gemeinsames Objekt erfolgt, in dem all diese Daten abgespeichert werden, ist es nötig, genau zu wissen, ob die Daten aktualisiert wurden. Zu diesem Zeitpunkt sind dann die bisher in diesem Objekt gehaltenen Daten ungültig und an ihre Stelle werden die neuen Daten geschrieben. Hierzu ist es wichtig, daß keine Inkonsistenzen auftreten können, d.h. Fälle, in denen im die Daten kapselnden Objekt sowohl Daten des vorherigen Zyklus als auch Daten des aktuellen Zyklus auftreten. Es muß außerdem dafür Sorge getragen werden, daß die Tatsache, daß ein neuer Satz Daten zur Verfügung steht, nach außen bekannt wird. Eine die Schnittstelle verwendende Applikation weiß dann, daß neue Daten zur Verfügung stehen und kann auf diese zugreifen. Nach obigen Überlegungen würde der Zugriff dann indirekt über das die Daten kapselnde Objekt stattfinden.
- Der Zugriff auf die meisten bekannten GPS-Empfänger erfolgt über eine serielle (RS 232) Schnittstelle. Es ist nötig, den Anschluß eines Empfängers für mehrere dieser seriellen Schnittstellen anzubieten, da sonst Probleme entstehen könnten, wenn eine bestimmte Schnittstelle bereits durch ein anderes externes Gerät besetzt ist. Wichtig ist in diesem Zusammenhang auch, daß die Wahl des Parameters „Baudrate“ flexibel gestaltet wird. Es hat sich zwar im Großen und Ganzen eine Baudrate von 4800 Baud als unterstützter Standard herauskristallisiert, aber verschiedene Empfänger bieten auch mögliche Baudraten an, die sowohl unter als auch über 4800 Baud liegen, wie z.B. 2400 oder 9600 Baud. Die Schnittstelle sollte einem Benutzer erlauben, flexibel auf eventuell auftretende Randbedingungen zu reagieren. Ein Beispiel wäre z.B. eine die Schnittstelle verwendende Anwendung, deren Zweck es ist, möglichst viele Positionsdaten in möglichst kurzer Zeit

zu sammeln. Bietet der verwendete Empfänger eine höhere Baudrate als die üblichen 4800 Baud an, so kann der Benutzer diese, sofern implementiert, voll ausnutzen.

4.2 Architektur der Schnittstelle

Die oben erwähnten Vorgaben und die Randbedingungen, die durch die besonderen Eigenschaften des verwendeten Protokolls und der allgemeinen Eigenschaften gängiger GPS-Empfänger bestimmt wurden, führten schließlich zusammen mit einigen anderen Überlegungen zu einem konkreten Entwurf der Schnittstelle. In diesem Abschnitt werden nun diese Überlegungen und die daraus entwickelte Architektur beschrieben.

4.2.1 Die weitergehenden Überlegungen

Nach der gefaßten Entscheidung, die Schnittstelle modular aufzubauen, war es nötig, sich Gedanken über den eigentlichen Grundriß der Schnittstelle zu machen.
Es wurde daraufhin ein Grundmodell entworfen, das in der späteren Implementierung umgesetzt wurde. Dafür war es zunächst klar, daß eines der oben angesprochenen Module alle Funktionen enthält, die zum Erhalt von allgemeinen NMEA 0183 Rahmen von einem Empfänger dient. Dieses Modul sollte auch als Grundmodul dienen, das als Basis für gegebenenfalls über den Umfang dieser Arbeit hinausführende Erweiterungen dienen soll. Eine denkbare Erweiterung wäre z.B. das Hinzufügen eines neuen Moduls, in dem die von einem bisher nicht berücksichtigten Hersteller von GPS-Empfängern definierten proprietären Rahmen und ihre Auswertung beschrieben werden. Im nächsten Kapitel wird eine einfache Erweiterung anhand eines kleinen Beispiels kurz skizziert. Beim Erstellen der Schnittstelle wurde ein solches Modul beispielhaft entworfen, es beschreibt, wie die proprietären Rahmen des Herstellers des Empfängers, der zum Test zur Verfügung gestellt wurde, verarbeitet werden sollen. In diesem Modul wird auch die Funktionalität bereit gestellt, die etwa für eine Änderung der Baudrate, oder des geodätischen Datums am Empfänger benötigt wird. Abbildung 15 skizziert dieses so entstandene Grundmodell:

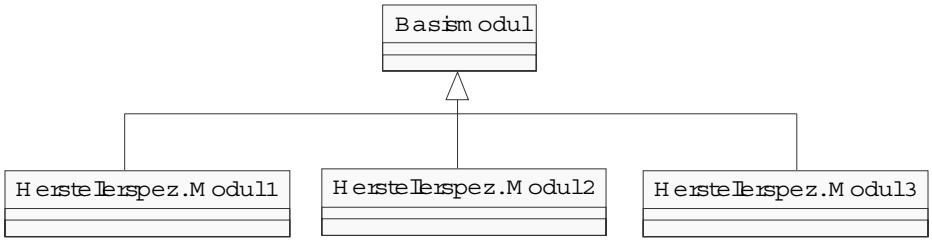


Abbildung 15: UML-Diagramm des ersten Entwurfs

Dieses Grundmodell, hier erweitert auf drei herstellerabhängige Module, war die Basis für weitergehende Verfeinerungen und Überlegungen.
Darauffolgend mußte die Eignung der vorgegebenen Programmiersprache Java 2 für die Implementierung des gewählten Grundmodells geprüft werden. Diese bekanntermaßen stark objektorientierte Sprache der Firma Sun legte die Verwendung des objektorientierten Paradigmas nahe. Dies deckte sich gut mit dem gewählten modularen Ansatz, der sich ohne größeren Aufwand in eine Klassenhierarchie mit Vererbung übertragen läßt. Im wesentlichen wurde darauf für jedes Modul eine eigene Klasse entworfen, wobei das Grundmodul als Vaterklasse sämtlicher Herstellermodule gedacht war. Im Zuge dieser Feststellung wurde entschieden, das Grundmodul als abstrakte Klasse zu entwickeln. Im objektorientierten

Sprachgebrauch bedeutet „abstrakt“, daß von dieser Klasse niemals eine direkte Instanz erzeugt wird und werden darf. Klassen dieser Art dienen vor allem zum Zusammenfassen gemeinsamen Codes. Im Gegensatz zur Funktionalität eines *Interfaces* lassen diese Art von Klassen aber neben Methodenköpfen, deren Implementierung von den erbenden Klassen übernommen werden muß, auch voll implementierte Methoden zu. Somit wurden in der Basisklasse die Methoden implementiert, die von allen erbenden herstellerspezifischen Unterklassen verwendet werden würden, wie z.B. die Verarbeitung der allgemeinen NMEA 0183 Rahmen. Lediglich deklariert wurden dagegen die Methoden, die nicht allgemeingültig zu implementieren waren, aus Entwicklersicht aber von jeder die Anbindung eines Empfängers realisierende Klasse angeboten werden sollten. Neu hinzugefügte Klassen, die die Funktionalität eines bisher nicht verwendeten Empfängertyps implementieren und hierfür zum Behandeln allgemeiner Funktionen von der Grundklasse erben, müssen demnach die verschiedenen in der Grundklasse deklarierten Methoden implementieren. Als erstes kleines Beispiel sei hier z.B. eine Methode erwähnt, die bei Aufruf den Typ des Empfängers zurückgibt. Die Existenz dieser Methode in jeder Unterklasse ist dann durch die Deklaration in der Oberklasse obligatorisch.
Als direkte Konsequenz aus dem Entwurf der Grundklasse als „abstrakt“ ergab sich die Notwendigkeit nach einer weiteren Klasse, die das bisher vorgestellte Grundmodell erweiterte. Da die Schnittstelle unabhängig vom angeschlossenen Gerätetyp benutzbar sein sollte, mußte diese allgemeine Funktionalität in einer zusätzlichen Klasse implementiert werden. Eine direkte Verwendung der Grundklasse schied durch die Definition einer als „abstrakt“ deklarierten Klasse ja aus. Als Folge daraus ergab sich folgendes abgeändertes Grundmodell (hier dargestellt mit nur 2 herstellerabhängigen Modulen). Die neu hinzugekommen Klasse wurde nach ihrer Funktionalität „GenericNMEASensor“ genannt.

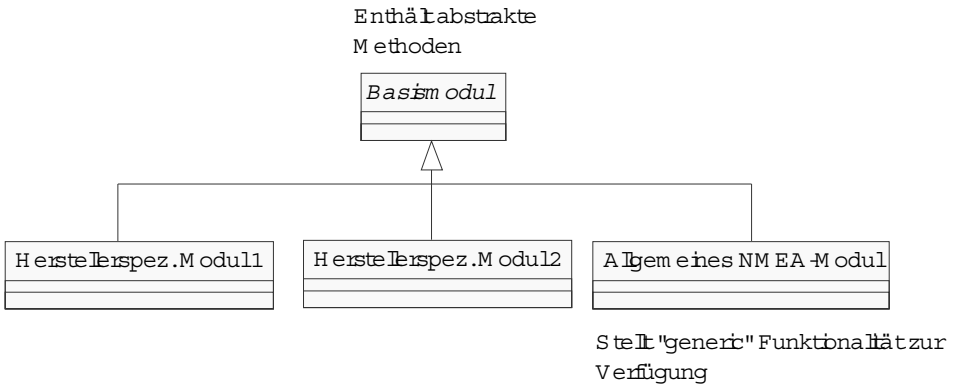


Abbildung 16: UML-Diagramm des erweiterten Entwurfs

Als nächstes mußten Überlegungen zum Umfang der geforderten Funktionalität angestellt werden. Die Erfüllung dieser Forderungen durch bereits implementierte und noch zu implementierende Unterklassen der Grundklasse wird u.a. durch die Deklaration entsprechender Methoden der Grundklasse als „abstrakt“ garantiert. Es wurden dabei folgende rudimentäre Punkte gefunden:
Ein Benutzer der Schnittstelle soll die Möglichkeit haben:

- Wählen zu können zwischen der Behandlung des angeschlossenen Empfängers als „spezieller Empfänger“, wenn die Auswertung der von diesem Empfänger gesendeten proprietären Rahmen bereits in einer eigenen herstellerspezifischen Unterklasse implementiert wurde, oder als „allgemeiner Empfänger“, wenn dies noch nicht geschehen ist. Eine Behandlung eines Empfängers als „allgemeiner Empfänger“ kann dabei unter Umständen zu einer Einschränkung des angebotenen Funktionsumfangs führen, denn wie

weiter oben erwähnt ist der Umfang der Information, der durch das allgemeine NMEA 0183 Protokoll erhalten werden kann, beschränkt.
Diese Forderung wird implizit durch die Möglichkeit eines Benutzers, eine Instanz einer beliebigen implementierten Klasse zu erzeugen, erfüllt. Der die Schnittstelle verwendende Benutzer entscheidet sich hier bereits bei der Erzeugung, als welcher Empfängertyp der von ihm angeschlossene Empfänger im Verlauf gelten soll.

- Ein einmal erschaffenes Empfängerobjekt zu starten, d.h. in einen Zustand zu versetzen, in dem Navigationsdaten von diesem Objekt erhalten werden können.
- Ein einmal erschaffenes Empfänger-Objekt zu stoppen, d.h. in einen Zustand zu versetzen, in welchem keine Navigationdaten mehr eintreffen, bzw. verarbeitet werden müssen.
- Sich den softwaremäßigen Typ des aktuell verwendeten Empfängers anzeigen zu lassen. Dieser Typ wird gemäß oben bereits bei der Erzeugung des Objekts festgelegt.

Dies sind die Grundanforderungen, die jede gemäß diesem Modell entworfene Erweiterung zu implementieren hat. Man kann sich dies vorstellen als die Reduktion der angebotenen Funktionalität eines Empfängerobjekts auf das allernotwendigste.
Durch Implementierung in der Grundklasse werden außerdem folgende Funktionalitäten für Empfängerobjekte beliebigen Typs verfügbar sein:

- Signalisierung, daß ein neuer Datensatz gesammelt wurde und zum Auslesen zur Verfügung steht.
- Zugriffsmöglichkeiten auf diesen Datensatz.
- Möglichkeit des Direktzugriffs auf ausgesuchte Daten eines Datensatzes, die durch das allgemeine NMEA 0183 Protokoll zu erhalten sind. Der Zugriff auf diese Daten kann also ohne die Indirektion über den kompletten Datensatz durchgeführt werden.
- Zugriff auf sämtliche Rahmen, die für die Erstellung des aktuellen Datensatzes empfangen wurden.
- Respezifikation des Ports, an dem der Empfänger angeschlossen ist.
- Anzeige, ob das Empfängerobjekt gerade läuft, d.h. aktiv Daten sammelt, oder sich gerade in Ruhestellung befindet.

Über die oben definierten Grundanforderungen hinaus bietet die die Anbindung des zur Verfügung gestellten Empfängers realisierende Klasse beispielhaft mehr Funktionalitäten als in den Grundforderungen verlangt wird an. Diese zusätzlichen Funktionalitäten können als Beispiel und Anstoß verstanden werden, um auch in neu zu implementierenden Klassen realisiert zu werden. Durch diese zusätzlichen Möglichkeiten werden die weiter oben beschriebenen wünschenswerten Anforderungen im wesentlichen erfüllt. Im einzelnen sind dies:

- Die Möglichkeit der Baudratenänderung auch im laufenden Betrieb, d.h. nicht nur über eine Parameterangabe beim Erzeugen des Empfängerobjekts.
- Das Simulieren eines Empfängers dieses Typs über eine anzugebende Datei. Bei dieser Datei handelt es sich um eine Textdatei, in der NMEA 0183 konforme Rahmen und/oder proprietäre Rahmen abgespeichert sind. Diese Art von Datei kann z.B. durch serielles Einfügen der empfangenen Rahmen in eine Datei erzeugt werden. Genaueres über dieses Thema wird später bei der Erläuterung der Beispielsanwendungen zu finden sein.
- Das Ändern des für die Positionsangabe notwendigen geodätischen Datums (siehe hierzu auch unter 2.2). Genau wie die Änderung der Baudrate ist es nötig, diese Funktionalität in einer produktspezifischen Klasse anzubieten, da nur hier die vom Gerät angebotenen Baudraten und geodätische Data bekannt und damit auf Gültigkeit überprüfbar sind. Näheres zu den angebotenen Möglichkeiten der Positionsbeschreibung wird unter 4.2.3 besprochen werden.

- Die Möglichkeit, eine begrenzte Auswahl der zu verwendenden Rahmen zu treffen. Diese Möglichkeit muß ebenfalls in einer spezifischen Klasse implementiert werden, da nur hier die möglicherweise vorkommenden Rahmen und ihre Bezeichnung bekannt sein können.
- Die Möglichkeit des Direktzugriffs auf bestimmte, wichtige Daten eines Datensatzes, die nicht allgemeiner Natur sind, die also aus proprietären Rahmen extrahiert wurden.
- Abfrage verschiedener Informationen, wie z.B. die gerade softwareseitig eingestellte Baudrate und das verwendete geodätische Datum.

Nachdem die Grundzüge der Architektur der Schnittstelle nun dargestellt wurden, muß noch näher auf einen anderen fundamentalen Teil der erstellten Arbeit eingegangen werden. Für eine die Anbindung eines GPS-Empfängers realisierende Architektur ist es nötig, Klassen für Positionsangaben bereitzustellen. Die diesen Teil der Arbeit betreffenden Entwurfsentscheidungen werden im folgenden Unterkapitel dargestellt.

4.2.2 Die Klassenbibliothek für die Positionsangabe

Diese Klassenbibliothek basiert auf den von Ralf Wagner an der Universität Stuttgart im Rahmen des Nexus-Projekts erstellten sogenannten “Location Klassen”. Eine Untermenge dieser Klassen findet sich, an den bisherig erstellten Schnittstellenentwurf angepaßt, in der entworfenen Klassenbibliothek wieder.

Eine wichtige Entwurfsentscheidung, die getroffen werden mußte, war das zu wählende Positionierungsmodell. Erst unter Beziehung auf ein solches Modell ist eine Positionsangabe möglich. In die Wahl kamen dabei *Earth Centered/Earth Fixed, geografische und konforme* Koordinatensysteme. Genaueres über die genannten Systeme ist in 3.1 zu finden. Da aus Zeit- und Aufwandsgründen nicht alle drei Alternativen berücksichtigt werden konnten, fiel die Wahl auf die Implementierung des auf *geografische* Koordinatensysteme basierenden Modells. Die Gründe, die diese Entscheidung herbeiführten sind hier angestellt.

- Geografische Koordinaten werden in den meisten GPS-Empfängern auch intern verwendet. Das bedeutet unter anderen, daß für Positionsangaben, die diesem Modell entsprechen, die vom GPS-Empfänger gelieferten Positionsinformationen direkt nutzbar sind. Dadurch kann auf eine Konvertierung oder Transformation der rohen Positionsinformation verzichtet werden, was sowohl der Komplexität des Moduls, als auch der der erzielten Genauigkeit der Positionsangabe zugute kommt. Dies hat seinen Grund im Fehlen der sonst bei jeder Koordinatentransformation auftretenden Verschlechterung der Positionsschärfe, die wiederum auf Rundungsungenauigkeiten und anderen notwendigen mathematischen Näherungen beruht.
- Geografische Koordinaten sind wohl die am meisten bekannte Art der Positionsangabe. Die meisten Menschen können sich unter einer Angabe der Art **48° nördlicher Breite, 9° östlicher Länge** etwas vorstellen, oder sind zumindest einmal etwas vergleichbarem begegnet. Geografische Koordinaten werden schon seit langer Zeit für den Zweck der Positionsangabe benutzt und es existieren eine Vielzahl von bereits gemäß diesem Modell vermessenen Punkten auf der Erdoberfläche. So gibt es z.B. die Möglichkeit, über das World Wide Web auf Datenbanken zuzugreifen, in denen die genauen Koordinaten einzelner Wohnhäuser in Städten wie Chicago abgelegt sind. Durch die Verwendung der geografischen Koordinaten in dieser Arbeit können erlangte Positionsangaben so direkt mit einer großen Anzahl von anderen Angaben verglichen und eingeordnet werden, was die Verwendbarkeit der gesammelten Daten natürlich beträchtlich erhöht.

- Die geografischen Koordinaten sind durch ihre mittlere Position im “Transformations-Schema” für Positionierungsmodelle (vgl. 3.1.5) die ideale Ausgangsposition für eventuell anfallende Koordinatentransformationen. Dies bedeutet, daß ausgehend von den geografischen Koordinatensystemen eine Transformation sowohl zu Earth Centered/Earth Fixed Koordinaten als auch zu konformen Koordinaten in einem Zug möglich ist, was wiederum der erzielten Positionsschärfe zu Gute kommt.

Weitere Entwurfsentscheidungen und ihre Gründe waren:

- Um einen Benutzer bei der Eingabe einer geografischen Koordinate, deren geografische Breite und Länge empfängerintern meist als Fließkommazahl gespeichert und ausgegeben wird, von einer Konvertierung vom gebräuchlichen ddd°mm’ss.ss’’ Format zu entlasten, wurde die Erweiterung der Klassenbibliothek zur Positionsangabe um einen zusätzlichen Datentyp beschlossen. Dieser Datentyp ermöglicht es einem Benutzer, eine auf die geografischen Koordinatensysteme basierende Positionsangabe direkt anzugeben. Eine Eingabe einer Position ist z.B. dann nötig, wenn die Distanz zwischen der aktuellen Position und einer bekannten Referenzposition berechnet werden soll. Dies wird später noch konkret erläutert, denn eine solche Funktionalitätsanforderung wurde an die die Schnittstelle verwendende Beispielsapplikation gestellt.
- Der von Ralf Wagner gewählte, auf Erweiterbarkeit ausgelegte Ansatz wurde beibehalten, es wurde daher mit Hilfe abstrakter Klassen eine Struktur geprägt, die von diese Architektur um andere Positionierungsmodelle erweiternde Klassen berücksichtigt werden muß. Als Folge ist garantiert, daß bei einer eventuellen Erweiterung die geforderten Funktionalitäten vorhanden sein werden. Ein Beispiel ist hier die Funktionalität *Bestimmung des Abstandes zu einem bestimmten Punkt* zu nennen, deren grundlegende Methode “distanceTo” in jeder ein Positionierungsmodell implementierenden Klasse vorhanden sein muß. Einzelheiten werden im folgenden Kapitel, das die eigentliche Implementierung aller Klassen beschreibt, angegeben.

5 Umsetzung des Entwurfs

In diesem Kapitel wird die eigentliche Implementierung der Schnittstelle beschrieben. Begonnen wird dazu mit grundlegenden Fragen, wie z.B. die Ansteuerung der seriellen Ports durchgeführt wurde, oder welchen Typ/Hersteller der verwendete Referenzempfänger hatte und wie sich dies auf die Menge der implementierten Datenrahmen auswirkte. Anschließend wird die entwickelte Bibliothek mittels eines UML-Diagramms veranschaulicht und es wird näher auf die Bedeutung und Verwendung ausgesuchter Objekte und ihrer Zugriffsmethoden eingegangen. Sollte es für das Verständnis der Entwicklung einer solchen Schnittstelle nötig sein, auch die für einen späteren Benutzer unsichtbaren internen Objekte und Methoden zu erläutern, so wird dies im folgenden getan. Ein kurzes Unterkapitel über durchgeführte Tests und eine Anleitung für eine eventuelle Erweiterung der Arbeit schließt diesen Teil ab.

5.1 Technische Grundlagen

5.1.1 Ansteuerung der seriellen Schnittstellen

Diese Arbeit wurde, wie schon weiter oben erwähnt in *Java 2* implementiert. *Java 2* bietet für sich alleine gestellt, keine Möglichkeit auf hardwareabhängige Ressourcen, wie sie z.B. die seriellen Schnittstellen sind, zuzugreifen. Dieses Problem kann jedoch durch die zusätzliche Installation der frei von http://www.java.sun.com erhältlichen, hardwareabhängigen „Java Communications“-API beseitigt werden. Dieses, oft auch als „javax.comm“-API bezeichnete Paket bietet eine umfassende Unterstützung des Zugriffs auf die seriellen (RS 232) und parallelen Schnittstellen (IEEE 1284) eines Rechners. Zum gegenwärtigen Zeitpunkt ist die „Java Communications“ –API allerdings nur für die Plattformen Win32 und Solaris erhältlich. Doch da eine Vorgabe dieser Arbeit die Lauffähigkeit unter MS-Windows war, reicht dies aus, um für diese Aufgabe geeignet zu sein. Allerdings muß als Konsequenz der Verwendung der „Java Communications“-API der Zugriff auf diese zur Laufzeit gewährleistet sein, d.h. die API muß auf einem Rechner auf dem die entwickelte Arbeit verwendet wird, installiert und ihre Position im Dateisystem bekannt sein.

5.1.2 Der zur Verfügung gestellte GPS-Empfänger

Für diese Arbeit wurde als Testgerät der *Garmin 25LP* DGPS-Empfänger zur Verfügung gestellt. Dabei handelt es sich um ein sogenanntes OEM-Board, d.h. einen GPS-Empfänger ohne weitergehende Funktionalität, wie z.B. einem grafischen Display oder der Möglichkeit zur Verwaltung von Wegpunkten usw. OEM-Boards bieten dementsprechend nur die Grundfunktionalität, die von einem GPS-Empfänger erwartet werden kann, nämlich die Ausgabe von Rohdaten für verschiedenste Navigations- und Positionierungsanwendungen und eine nahezu atomgenaue Uhr. Der *Garmin 25LP* Empfänger verfügt über 12 Kanäle, d.h. er kann die Position theoretisch aus den Daten von bis zu 12 Satelliten bestimmen. Tatsächlich wird bei diesem Empfänger die Position aber nur aus maximal 8 Satelliten trianguliert. Durch einen angeschlossenen Langwellenempfänger ist er in der Lage, die Vorteile des DGPS-Systems auszunutzen und erreicht damit eine hohe Genauigkeit bei der Positionsbestimmung. Die Datenaus- und Eingabe findet gemäß dem in NMEA 0183 spezifizierten Standard statt, zusätzlich werden von der Firma Garmin definierte proprietäre Rahmen verwendet. Vom Garmin25LP unterstützte Baudraten für die Datenübertragung über die serielle Schnittstelle sind z.B. 4800 und 9600 Baud, es werden aber noch einige andere angeboten. Ein vom Empfänger unterstütztes geodätische Datum ist neben vielen anderen z.B.

das bekannte WGS-84. Im folgenden werden nun die in dieser Schnittstelle verwendeten allgemeinen und proprietären Rahmen genannt und deren einzelne Felder kurz vorgestellt. Begonnen wird hierfür mit einer Auflistung der verwendeten, vom Garmin25LP unterstützten allgemeinen Rahmen. Daran schließt sich eine Tabelle mit den verwendeten, unterstützten Garmin-proprietären Rahmen an. Im Verlauf der Realisierung wurde entschieden, die Behandlung einiger zusätzlicher allgemeiner NMEA 0183 Rahmen, die nicht vom Garmin 25LP unterstützt werden zu implementieren. Der Grund für diese Entscheidung liegt in einer verbesserten Verwendbarkeit begründet. Die Beschreibung dieser zusätzlichen Rahmen wird in einer dritten Tabelle dargestellt.

Name des Rahmens	**Bedeutung der einzelnen Felder**
$GPGGA (Global Positioning System Fix) *Dieser Rahmen enthält vor allem Daten, die eine für eine Klassifizierung einer Positionsbestimmung verwendet werden, d.h. unter anderem die Position selbst und deren Qualität.*	<1> UTC Zeit der Positionsbestimmung <2> geografische Breite der Position <3> Hemisphäre der geografischen Breite <4> geografische Länge der Position <5> Hemisphäre der geografischen Länge <6> Klassifizierung der Qualität der durchgeführten Positionsbestimmung (keine/GPS/DGPS-Korrektur durchgeführt) <7> Anzahl der Satelliten, die für eine Positionsberechnung verwendet wurden. <8> Größe der horizontalen Komponente der bei der Positionsbestimmung entstehenden Grauzone (HDOP). <9> Höhe über dem mittleren Meeresspiegel. <10> Höhe des zugrunde liegenden Geoids (3.1.1) an dieser Stelle. <11> Sekunden seit Empfang des letzten DGPS-Korrektursignals. <12> ID der DGPS-Referenzstation
$GPGSA (GPS DOP and Active Satellites) *Dieser Rahmen enthält vor allem Daten, die für eine genauere Auswertung der Umstände einer bestimmten Position benutzt werden können, z.B. alle räumlichen *DOP-Werte, oder die ID-Nummern der bei der Positionsbestimmung beteiligten Satelliten.*	<1> Betriebsmodus des Geräts (wird nicht verwendet). <2> Art der durchgeführten Korrektur der Positionsbestimmung (keine/2D/3D). <3> Pseudorange number (PRN) der für die Positionsbestimmung verwendeten Satelliten (bis zu 12 unterschiedliche Satelliten). <4> Größe der Positionierungsgrauzone (PDOP). <5> Horizontaler Anteil an der Positionierungsgrauzone (HDOP). <6> Vertikaler Anteil an der Positionierungsgrauzone(VDOP)
$GPGSV (GPS Satellites in View) *Dieser Rahmen enthält Daten über die bei der Positionsbestimmung verwendeten Satelliten.*	*Dieser Rahmen kann in mehreren Teilen (bis zu 3) gesendet werden, dabei werden die Felder <4> bis <7> ggf. bis zu 4 mal wiederholt.* <1> Insgesamte Anzahl der zu diesem logischem Rahmen gehörigen Teilrahmen. <2>Sequenznummer des aktuell gesendeten Teilrahmens <3> Gesamtanzahl der „sichtbaren“ Satelliten (bis zu 12) <4> PRN eines der „sichtbaren“ Satelliten <5> Erhebung des Satelliten relativ zum Empfänger (Elevation) <6> Größe des Azimut des Satelliten <7> Signalstärke des Satelliten
$GPRMC (Recommended Minimum Specific GPS/Transit Data)	<1> UTC Zeit der Positionsbestimmung <2> Informationsstatus (gültig/ungültig) <3> geografische Breite

Der in diesem Rahmen enthaltene Datensatz wird von Daten gebildet, die die NMEA als empfohlene Mindestinformation für die Navigation ansieht.	<4> Hemisphäre der geografischen Breite <5> geografische Länge <6> Hemisphäre der geografischen Länge <7> Geschwindigkeit über Grund (in Knoten) <8> Kurs über Grund <9> UTC Datum der Positionsbestimmung <10> Magnetische Variation <11> Richtung der magn. Variation (Ost/West)
$GPVTG (Track Made Good and Ground Speed with GPS Talker ID) *Dieser Rahmen kapselt die Informationen, die Geschwindigkeit und Kurs betreffen.*	<1> True course über Grund <2> Magnetic course über Grund <3> Geschwindigkeit über Grund (in Knoten) <4> Geschwindigkeit über Grund (in Km/h)

Tabelle 3 : Allgemeine vom Testempfänger unterstützte NMEA 0183 Rahmen

Die vom Empfänger unterstützten und in dieser Arbeit verwendeten proprietären Rahmen sind folgende:

$PGRMI (Sensor Initialization Information) *Dieser Rahmen wird dazu benutzt, den Empfänger zu initialisieren und führt u.a. auch einen empfängerseitigen Reset durch.*	*Dieser Rahmen kann sowohl zum Empfänger gesendet werden, als auch von diesem empfangen werden.* <1> geografische Breite <2> Hemisphäre der geografischen Breite <3> geografische Länge <4> Hemisphäre der geografischen Länge <5> UTC Datum <6> UTC Zeit <7> Kommandos für Empfänger (Position bestimmen/Reset durchführen)
$PGRMC (Sensor Configuration Information) *Dieser Rahmen wird u.a. dazu benutzt, das vom Benutzer gewünschte geodätische Modell zur Positionsangabe, oder die bevorzugte Baudrate am Empfänger einzustellen. Er stellt auch eine Eingabemöglichkeit für vom Benutzer selbst definierte geodätische Daten dar.*	*Dieser Rahmen kann sowohl zum Empfänger gesendet werden, als auch von diesem empfangen werden.* <1> Art der Korrektur (automatisch/nur 2D Korrektur/nur 3D Korrektur) <2> Höhe über mittlerem Meeresspiegel <3> Garmin-eigener Index des gewünschten geodätischen Datums. Wenn ein vom Benutzer selbst definiertes geodätisches Datum angegeben wurde (Index = 96), müssen Feld <4> bis <8> gültige Werte enthalten, ansonsten müssen diese Felder leer sein. <4> Länge der Haupthalbachse des vom Benutzer definierten Referenzellipsoiden. <5> Abflachung des vom Benutzer gewählten Referenzellipsoiden. <6> Verschiebung des Nullpunkts in x-Richtung bei Angabe einer EC/EF Koordinate <7> Verschiebung des Nullpunkts in y-Richtung bei Angabe einer EC/EF Koordinate <8> Verschiebung des Nullpunkts in z-Richtung bei Angabe einer EC/EF Koordinate <9> Wahl des gewünschten Ausgabemodus (alle Positionsangaben ausgeben (DGPS und GPS)/nur DGPS-Positionsangaben ausgeben. <10> Garmin-eigener Index der gewünschte Baudrate. <11> Wahl eines Filters für Geschwindigkeitsausgaben <12> Frequenz-Einstellung für evtl. PPS Modus
$PGRMO (Output Sentence Enable/Disable) *Dieser Rahmen wird dazu benutzt, bestimmte Rahmen*	*Dieser Rahmen kann sowohl zum Empfänger gesendet werden, als auch von diesem empfangen werden.*

zu aktivieren oder zu deaktivieren	<1> Bezeichner des zu modifizierenden Rahmentyps. <2> Angabe der Modifikation (deaktivieren/aktivieren/alle Rahmen deaktivieren/alle Rahmen aktivieren)
$PGRME (Estimated Error Information) *Dieser Rahmen beinhaltet Informationen, die den absoluten Fehler der Positionsmessung wiedergeben.*	<1> Berechneter horizontaler Positionsfehler <2> Berechneter vertikaler Positionsfehler. <3> Berechneter gesamter Positionsfehler.
$PGRMF (GPS Fix Data Sentence) *Dieser Rahmen ist eine erweiterte Version der empfohlenen Mindestinformation für die Navigation.*	<1> GPS Wochenindex <2> GPS Sekunden <3>UTC Datum der Positionsbestimmung <4> UTC Zeit der Positionsbestimmung <5> Stand des UTC „leap second“ Zählers <6> geografische Breite <7> Hemisphäre der geografischen Breite <8> geografische Länge <9> Hemisphäre der geografischen Länge <10> Modus (vom Benutzer bestimmt/automatisch, wird nicht verwendet) <11> Art der Positionsbestimmung (keine/2D/3D) <12> Geschwindigkeit über Grund (in km/h) <13> Kurs über Grund <14> gesamte Genauigkeitsabschwächung (PDOP) <15> Index der Zeitdifferenz am Empfänger, aus der die Genauigkeitsabweichung berechnet wird.
$PGRMT (Sensor Status Information) *Dieser Rahmen kann für Diagnosezwecke am Empfänger eingesetzt werden.*	<1> Produktbezeichnung, Modell und Firmwareversion des Empfängers <2> Ergebnis eines ROM-Checksumtests (OK/Fehler) <3> Überprüfung des Empfängers auf Fehler (OK/Fehler) <4> Verlust gespeicherter Daten (OK/Fehler) <5> Uhrendrift (OK/Fehler) <6> Quarzdrift (OK/Fehler) <7> Fehler beim Datensammeln (OK/Fehler) <8> Board-Temperatur (in Grad) <9> Board Koonfigurationsdaten verloren (OK/Fehler)
$PGRMV (3D Velocity Information) *Die in diesem Rahmen enthaltenen Daten schlüsseln die Geschwindigkeit in ihre drei Komponenen auf.*	<1> Ost-Komponente der Geschwindigkeit(in m/s). <2> West-Komponente der Geschwindigkeit(in m/s). <3> Vertkale Komponente der Geschwindigkeit (in m/s)

Tabelle 4: Garmin-proprietäre, vom Empfänger unterstützte Rahmen

Die oben erwähnten zusätzlich implementierten NMEA-Rahmen sind:

$GPHDM (Heading – Magnetic) *Dieser Rahmen liefert die Richtung (magnetisch), in die sich der Empfänger bewegt.*	<1> Magnetische Richtung des Empfängers
$GPRMB (Recommended minimum navigation information) *Dieser Rahmen enthält die empfohlene Mindestinformation, die bei Existenz mindestens eines aktiven Wegpunkts in einer eingegebenen Route zur Verfügung gestellt werden sollte.*	<1> Informationsstatus (gültig/ungültig). <2> Abweichung vom vorberechneten Kurs zum nächsten Wegpunkt (Seemeilen). <3> Richtung, in die zur Korrektur gesteuert werden muß (Links/Rechts). <4> ID des Wegpunktes, der Ausgangspunkt der gerade befahrenen Teilstrecke ist. <5> ID des Wegpunktes, der Zielpunkt der gerade befahrenen Teilstrecke ist. <6> Geografische Breite oben genannten Wegpunkts. <7> Hemisphäre der geografischen Breite des oben genannten Wegpunkts.

	<8> Geografische Länge des oben genannten Wegpunkts. <9> Hemisphäre der geografischen Länge des oben genannten Wegpunkts. <10> Entfernung zum oben genannten Wegpunkt (Seemeilen). <11> Richtung des oben genannten Wegpunkts (true). <12> Geschwindigkeitskomponente in Richtung des oben genannten Wegpunkts. <13> Signalisierung, ob oben genannter Wegpunkt bereits erreicht wurde (A), oder noch nicht erreicht worden ist (V).

Tabelle 5 : Zusätzlich implementierte allgemeine NMEA 0183 Rahmen

5.1.3 Der Xybernaut Mobile Assistant

Wie schon früher erwähnt, war eine weitere Anforderung an die Schnittstelle die Verwendbarkeit mit einem sogenannten *Wearable Computer* der Firma Xybernaut. Ein solcher Rechner hat die Maße, das Gewicht und die Form, um am Körper getragen zu werden. Zusätzlich verfügt er über Ein- und Ausgabemöglichkeiten, die es ihm ermöglichen, auch in beengten, für die Arbeit an einem „normalen“ Rechner ungeeigneten Plätzen bedienbar zu sein, z.B. ein druckempfindlicher Bildschirm für die Ein- und ein vor ein Auge plazierbarer Minimonitor für die Ausgabe. Eine mögliche Anwendung wäre z.B. die Flugzeugfertigung, bei der für Arbeiten im Rumpf oft Baupläne zu Rate gezogen werden müssen. Diese können dann auf einem Xybernaut Mobile Assistant schnell zur Verfügung gestellt werden. Nähere Angaben über die Xybernaut Technologie sind auf http://www.xybernaut.com zu erfahren.

5.2 Die Umsetzung des Entwurfs

Hier wird nun die konkrete Realisierung der gestellten Aufgabe beschrieben. Dazu soll ein UML-Diagramm der Gesamtarchitektur vorangestellt und die einzelnen Komponenten der Schnittstelle nach und nach genauer erklärt werden. Zu einer detaillierten Erklärung der einzelnen Komponenten gehört ein UML-Diagramm und eine Erläuterung der wesentlichen Bestandteile und Abläufe innerhalb dieser Klasse. Auf die Verknüpfung der einzelnen Komponenten untereinander und wie diese zusammenarbeiten wird ggf. ebenfalls eingegangen.

Die entwickelte Schnittstelle besteht aus folgenden Komponenten (im Java-Gebrauch auch als „package“ bezeichnet), im Anhang werden die Signaturen der einzelnen Methoden angegeben:

- Dem Package „Sensor“, in dem die eigentliche Funktionalität der Schnittstelle gekapselt ist. In diesem Package befinden sich u.a. alle anderen Packages. Hier befindet sich auch die die Navigationsdaten kapselnde Klasse „GPSData“.
- Dem Package „Gps“. Hier werden Klassen gekapselt, die für die Anbindung eines speziellen oder eines allgemein verwendbaren GPS-Empfängers nötig sind. In diesen Klassen wird z.B. die Auswertung empfangener NMEA Rahmen, oder die Steuerung des angeschlossenen Empfängers behandelt.

- Dem Package „Location“. Wie weiter oben bereits angemerkt, realisiert diese Komponente den neuen Datentyp „Positionsangabe“. Es werden hier Klassen zur Verfügung gestellt, mit deren Hilfe aus den rohen Positionsdaten des Empfängers gültige, verwaltbare Positionsangaben bestimmt werden können.

Zusätzlich zu diesen existieren noch andere Komponenten. Teils wird auf diese noch zurückgekommen werden (wie z.B. auf das Package „Gui“, in dem die grafische Benutzeroberfläche des im nächsten Kapitel beschriebenen, die Schnittstelle verwendenden Utility „GPSApplication“ beschrieben wird).

5.2.1 Das Package „Sensor“

Das Package „Sensor“ ist das Hauptmodul der entwickelten Schnittstelle. Alle anderen, die Funktionalität der Schnittstelle ausmachenden Packages und Klassen, sind hier zu finden. Nachfolgend wird die sich hier befindliche Klasse „GPSData“, die zur Kapselung der vom GPS-Empfänger erhaltenen Navigationsdaten verwendet wird, genauer erläutert.

Die Klasse „GPSData“:

Ziel der Einführung dieser Klasse war es, alle in einem Rahmenzyklus gesammelten Positions- und Navigationsinformationen in einem gemeinsamen Objekt zu kapseln. Diese Klasse wurde daher so aufgebaut, daß jeder von dieser Schnittstelle unterstützten Navigations- oder Positionsinformation eine eigene Variable zugeordnet wird, in die diese Information bei Erhalt geschrieben wird. Der spätere Zugriff auf diese so gespeicherte Information erfolgt aus Gründen des „Data Hiding“ über definierte Schnittstellen, so daß unbeabsichtigtes Zugreifen und Verändern ausgeschlossen wird. Konkret besagt dies, daß jeder nach außen unsichtbaren Variablen zwei Methoden zugeordnet wurden, eine „get“-Methode, mit der der Wert der Variablen abgefragt werden kann und eine „set“-Methode, mit der der Wert einer Variablen gesetzt werden kann. Mögliche Datentypen für die Variablen sind „String“ (z.B. Zeit und Datum), „Integer-Wert“ (z.B. Indices und bestimmte Zahlenwerte), „Fließkomma-Wert“ (z.B. bestimmte Zahlenwerte), „Character“ (z.B. bestimmte Flags), oder auch Felder von einem der vorherigen Variablentypen (z.B. eine Menge von logisch zusammengehörigen Werten). Die verschiedenen Variablen sind außerdem noch logisch in zwei Gruppen aufgespalten worden, zur einen gehören dabei alle Variablen, die Daten repräsentieren, die durch allgemeine NMEA 0183 Rahmen zu erhalten sind. Die andere Gruppe besteht aus Variablen, deren Werte nur durch herstellerproprietäre Rahmen zu beziehen sind. Diese Unterscheidung ist wichtig, denn bei ausschließlicher Verwendung allgemeiner Rahmen kann damit nach Ende eines Rahmenzyklus auf das Rücksetzen der Variablen der zweiten Gruppe auf Defaultwerte verzichtet werden. Durchgeführt wird die Unterscheidung mittels der Methode „resetAllValues(boolean)“, deren Parameter angibt, ob die Variablen beider Gruppen, oder nur die Variablen der „allgemeinen“ Gruppe zurückgesetzt werden sollen. Dieser Vorgang wird in den kommenden Abschnitten noch einmal angesprochen werden. Wie bereits eingangs erwähnt, werden die Signaturen der einzelnen Methoden im Anhang angegeben.

5.2.2 Das Package „Gps“

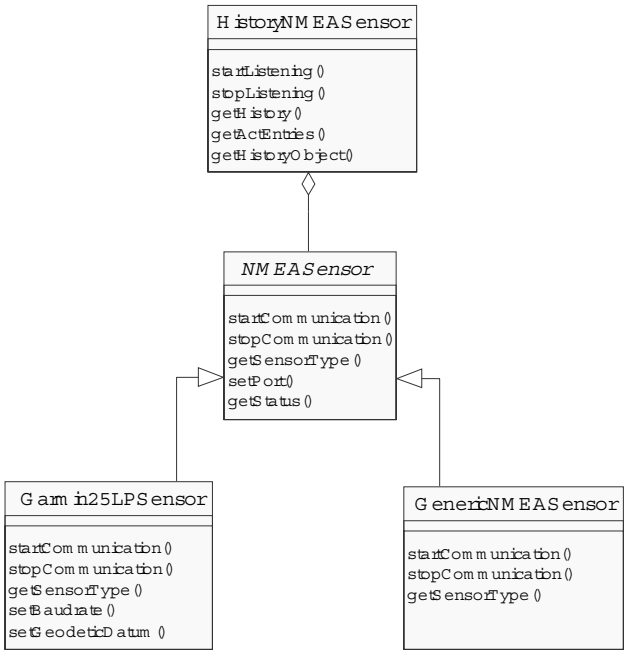


Abbildung 17: UML-Diagramm des "GPS"-Packages

In dieser Komponente werden all die Klassen gekapselt, die für die Anbindung und den Erhalt von Daten eines GPS-Empfängers benötigt werden. Wie im Entwurf entschieden wurde, wurden hierfür mehrere verschiedene Klassen benötigt, um z.B. die Behandlung der verschiedenen Rahmensätze der einzelnen Hersteller der Empfänger zu erledigen. Die einzelnen bei dieser Arbeit erstellten Klassen des „Gps“ Packages, deren Aufbau und die wichtigsten Abläufe werden im folgenden dargestellt.

Die Klasse „NMEASensor“

Diese Klasse ist die Oberklasse aller die eigentliche Anbindung eines Empfängers eines bestimmten Typs realisierenden Klassen. Sie wurde als „abstrakt“ deklariert, um durch Angabe einiger abstrakter Methoden die Implementierung dieser in den Unterklassen obligatorisch zu machen. Auf diese Art wird einer Unterklasse eine einheitliche Struktur aufgeprägt, dies gilt auch für eventuell spätere Erweiterungen der Klassenbibliothek. Um die Vorteile der Vererbung gut auszunutzen, wurde beim Entwurf und der Implementierung der Klassen dieses Packages darauf geachtet, allgemeine, für alle Klassen geltende Methoden und Variablen in dieser Oberklasse zu plazieren. Auf diese Weise wird eine spätere Erweiterung erleichtert, denn in der Vaterklasse befindliche, nicht private Methoden und Variablen stehen in jeder von ihr erbenden Unterklasse zur Verfügung. Beispielsweise werden die Input/Output-Streams, die für eine Kommunikation über eine serielle Schnittstelle benötigt werden, in dieser Klasse deklariert, denn diese können allgemeingültig von jeder Unterklasse verwendet werden. Weiterhin wurden natürlich alle Methoden, die für die Auswertung der allgemeinen NMEA 0183 Rahmen benötigt werden, hier implementiert. Wie weiter oben schon erwähnt, senden die meisten GPS-Empfänger ihre Daten kontinuierlich, d.h. ohne merkbare Pause zwischen den einzelnen Zyklen.

Es wurde auch schon angesprochen, daß dadurch die Notwendigkeit der Festlegung eines eindeutig erkennbaren Rahmens entsteht, an dessen Erhalt das Ende eines Rahmenzyklus erkannt werden kann. Da dieser Rahmen von möglichst vielen Empfängern unterstützt werden soll, fiel die Wahl auf den Rahmen „*Recommended Minimum Specific GPS/TRANSIT Data*“, der auch kurz mit *$GPRMC* bezeichnet wird. Dieser Rahmen wurde, wie schon aus dem Namen ersichtlich ist, als Träger einer empfohlenen Mindestinformation entwickelt, in seinen Datenfeldern werden die (aus Sicht der *NMEA*) wichtigsten Informationen transportiert. Dadurch ist die Wahrscheinlichkeit groß, daß dieser Rahmen von den meisten Empfängern unterstützt wird. Da die Übernahme der im Laufe eines Zyklus gesammelten Daten erst nach Auswertung dieses Rahmens geschieht, ist garantiert, daß bei Verwendung der Schnittstelle mit einem diesen Rahmen unterstützenden Empfänger mindestens die in diesem Rahmen befindlichen Daten zur weiteren Verarbeitung zur Verfügung stehen. Auf diese Weise stehen also auf jeden Fall die wichtigsten Daten zur Verfügung, unabhängig von den restlichen unterstützten Rahmens eines Empfängers.
Die bei den Entwurfsfragen geforderte Signalisierung, daß ein Rahmenzyklus beendet wurde und neue Daten zur Übernahme bereit stehen, wurde, da allgemein verwendbar, ebenfalls in dieser Klasse implementiert. Es stehen nun für verwendende Anwendungen die beiden öffentlichen Methoden „getUpdated()“ und „setUpdated(boolean)“ zur Verfügung. Die erste der beiden Methoden dient der Abfrage, ob ein Zyklusende erreicht wurde und ist dafür gedacht, periodisch aufgerufen zu werden. Die zweite Methode ist dafür gedacht, den Wert der Variablen, die den aktuellen Zustand beschreibt, zu verändern. Zum Beispiel kann eine Anwendung nach Übernahme der im Laufe eines Zyklus in einem Objekt des Typs „GPSData“ angesammelten Daten mit Hilfe dieser Methode den aktuellen Zustand auf „Zyklusende noch nicht erreicht“ zurücksetzen. Auf diese Weise wird erst nach Neusetzen dieses Wertes auf „Zyklusende erreicht“ durch die Schnittstelle wieder auf das „GPSData“-Objekt zugegriffen.
Für den Zugriff auf dieses Objekt steht die öffentliche Methode „getCurrentGPSData()“ zur Verfügung. Diese liefert das die aktuell gültigen Daten enthaltende GPSData-Objekt zurück. Sollten die gesammelten Daten aus irgendeinem Grund nicht gültig sein, was z.B. zu Beginn einer Positionsbestimmung oftmals auftritt, dann wirft die Methode eine „NoValidValueException“, um dies einer Anwendung bekannt zu machen.

Die andere Art des Zugriffs auf bestimmte Daten ist die im Entwurf beschlossene Direktzugriffsmöglichkeit. Hier wird bspw. auf aktuelle Positionsdaten nicht über das diese enthaltende GPSData-Objekt zugegriffen, sondern mittels der Methode „getCoordinates()“. Es existieren noch andere Direktzugriffsmethoden, mit deren Hilfe einfacher auf die wichtigsten Positionsdaten zugegriffen werden kann, im einzelnen sind dies „getAccuracy()“ zur Bestimmung der Positionierungsunschärfen VDOP, HDOP und PDOP und „getFixQuality()“ für die Bestimmung der Qualität der Positionsbestimmung.
Mittels der Methode „getSentences()“ lassen sich alle Rahmen bestimmen, aus denen die bisherigen Daten extrahiert wurden. Die Methode „setPort(String)“ dient der Angabe der seriellen Schnittstelle, über die der Empfänger angeschlossen ist.
Der aktuelle Status eines Datensammlungslaufes kann durch Aufruf der Methode „getStatus()“ ermittelt werden, mögliche Werte sind dabei unter anderem die als Konstanten definierten Werte:

- „UNDEFINED“, dies ist der Defaultwert, er ist nur vor der Initialisierung gesetzt.
- „INITIALIZED“, dies ist der Fall, wenn ein die Anbindung eines Empfängers realisierendes Objekt erfolgreich initialisiert wurde, aber noch kein Datensammlungslauf gestartet wurde.

- „ERROR“, dies ist der Fall, wenn ein Fehler bei der Initialisierung eines Empfängerobjekts auftrat und das Objekt damit nicht bereit für weitergehende Verwendungen ist.
- „RUNNING“, dies signalisiert einen aktiven Datensammlungslauf, der gültige Daten zurückliefert.
- „STOPPED“, dies signalisiert, daß ein zuvor aktiver Datensammlungslauf angehalten wurde.
- „COMPUTING_VALID_POSITION_FIX“, dies signalisiert den Erhalt ungültiger Positionsinformationen, wie es z.B. nach dem Einschalten eines Empfängers bis zur initialen Positionsbestimmung vorkommen kann.

Die weiter oben vorgenommene Trennung der in der Klasse „GPSData“ vorhandenen Variablen in zwei logische Gruppen wird mit Hilfe der Variable „basicNMEA“ vorgenommen. Diese wird in den Konstruktoren der Klassen, die für die Anbindung von Empfängern, die auch proprietäre Rahmen benutzen, entstanden sind, auf den Wert *false* gesetzt. In Klassen, die ausschließlich die Behandlung allgemeiner NMEA Rahmen implementieren, erhält diese den Wert *true*. Anhand des Werts dieser Variablen ist es in der Klasse „GPSData“ dann möglich, die oben angesprochene logische Unterscheidung umzusetzen. Der konkrete Handlungsablauf bei Erreichen eines Zyklusendes und der nachfolgenden Übernahme der gesammelten Werte wird nachfolgend im Kapitel 5.3 unter „Ablauf der Datensammlung“ beschrieben.

Die Klasse „GenericNMEASensor“

Diese Klasse realisiert die Forderung nach der Verwendbarkeit der Schnittstelle mit Empfängern unterschiedlicher Hersteller. Es wurde weiter oben schon angesprochen, daß die Entwicklung dieser Klasse durch die Deklaration ihrer Oberklasse als „abstrakt“ nötig wurde. Mit Hilfe dieser Klasse können auch Empfänger mit der Schnittstelle verwendet werden, für die keine eigene Klasse implementiert wurde. Natürlich müssen bei einer derartigen Verallgemeinerung erhebliche Abstriche bei der angebotenen Funktionalität gemacht werden. Beispielsweise werden in dieser Klasse keine Methoden zum Ändern der Baudrate oder dem Umstellen des verwendeten geodätischen Datums angeboten, da bei einem solchen allgemeinen Ansatz keine Vorhersagen über tatsächlich unterstützte Baudraten und geodätische Data getroffen werden kann. Natürlich werden aber in dieser Klasse die in der Oberklasse durch Methodendeklaration spezifizierten Mindestanforderungen erfüllt, wie die Steuerung der Datensammlung (z.B. durch die Methoden startCommunication() und stopCommunication()). Für die Anbindung eines „Generic“-Empfängers wurde ein Defaultkonstruktor, der ohne Angabe von Parametern aufgerufen werden kann und die Baudrate auf 4800 Baud, das verwendete geodätische Datum auf WGS-84 und die serielle Schnittstelle, an der der Empfänger angeschlossen ist, auf „COM1“ setzt. Ein zweiter Konstruktor läßt dagegen die Wahl einer seriellen Schnittstelle durch Angabe eines dementsprechenden Parameters frei.

Die Klasse „Garmin25LPSensor“

Wie schon erwähnt, wurde für die Entwicklung dieser Arbeit ein Empfänger der Firma Garmin, genauer ein Garmin 25LP OEM-Board zur Verfügung gestellt. Diese Klasse realisiert nun die Anbindung dieses Empfängers an die bisher erstellte Schnittstelle. Es wäre natürlich auch möglich gewesen, auch diesen Empfänger als „generic“ zu betreiben, d.h. dessen Anbindung durch eine Instanz der Klasse „GenericNMEASensor“ zu realisieren. Doch dann hätte einerseits auf die Informationen verzichtet werden müssen, die nur über Garmin-

proprietäre Rahmen zu erhalten sind und andererseits wäre es so nicht möglich gewesen, z.B. die Baudrate oder das den Positionsangaben zugrunde liegende geodätische Datum zu ändern. Änderungen der Baudrate bzw. des geodätischen Datums werden über die dafür vorgesehenen Methoden „setBaudrate(int)“, bzw. „setGeodeticDatum(int)“ vorgenommen.
Um konsistente Einstellungen am Empfänger und am Rechner nach Ändern der Baudrate oder des geodätischen Datums zu erhalten, wird nach erfolgreicher Änderung einer dieser beiden Werte eine Textdatei auf die Festplatte geschrieben, in der die aktuellen Werte für Baudrate und Datum eingetragen sind. Die dort abgespeicherten Werte werden bei Neustart eines Datensammlungslaufes durch einen Aufruf der Methode „startCommunication()“ ausgelesen und z.B. für die Festlegung der Datenübertragungsparameter der seriellen Schnittstelle verwendet.
Wie in der Oberklasse stehen für den Zugriff auf bestimmte, logisch zusammengehörige Daten zusätzliche Methoden zum Direktzugriff bereit. Dies sind die Methoden „getError()“ und „getVelocity()“, die einerseits eine detaillierte Fehlerabschätzung des relativen Positionierungsfehlers, anderseits eine Liste der verschiedenen räumlichen Komponenten der aktuellen Geschwindigkeit zurückgibt.
In dieser Klasse wird ebenfalls die eigentliche Bedeutung der in der Oberklasse deklarierten Konstanten „FULL_DATA“ und „SIMPLE_DATA“ definiert, die zum Festlegen der zu verwendenden Rahmen dienen sollen. Die Bedeutung der beiden Konstanten für jede Unterklasse wird durch die Implementierung der privaten Methode „activateSentences(int)“ festgelegt, in diesem Fall werden für „FULL_DATA“ sämtliche unterstützte Rahmen freigeschaltet, im Fall von „SIMPLE_DATA“ die Rahmen $GPRMC, $GPGSA, $GPGGA, $GPVTG, $PGRME und $PGRMV. Für die Auswahl einer der beiden Möglichkeiten ist die Methode „setSentencesEnabled(int)“ vorgesehen.
Die im Entwurf vorgesehene Ausgabemöglichkeit der aktuellen Baudrate oder des geodätischen Datums, das den Positionsangaben zugrunde liegt, ist durch die Methoden „getCurrentBaudrate()“ bzw. „getCurrentGeodeticDatum()“ implementiert worden. Die Implementierung der bereits angesprochenen Möglichkeit der Simulation eines Empfängers wurde ebenfalls in der „Garmin25LPSensor“-Klasse vorgenommen. Dazu wurde ein zusätzlicher Konstruktor implementiert, dessen einziger Parameter eine Referenz auf eine Textdatei, in der die Simulationsdaten abgespeichert sind, darstellt. Für Einzelheiten über das Erzeugen und die Verwendung einer solchen Datei sei auch hier wieder auf das nächste Kapitel, das die Beispielsapplikationen behandelt, hingewiesen. Für die Anbindung eines realen Garmin 25 LP GPS-Empfängers stehen mehrere Konstruktoren bereit, deren Parameter z.B. die initial zu verwendende Baudrate, das geodätische Datum, oder auch die serielle Schnittstelle, an der der Empfänger angeschlossen ist, festlegen. Als Defaulteinstellung wurde die Baudrate auf 4800 Baud, das verwendete geodätische Datum als WGS-84 und die verwendete serielle Schnittstelle auf „COM1“ gesetzt.

Die Klasse „HistoryNMEASensor“

Diese Klasse wurde implementiert, um die bis dahin entstandene Architektur um die Möglichkeit zu erweitern, eine Historie über erhaltene GPS-Daten erstellen zu können. Dazu wurde beschlossen, daß die Klasse „HistoryNMEASensor“ auf die bereits erstellten, die Anbindung der verschiedenen Empfängertypen realisierenden Klassen aufsetzbar ist und so die bereits vorhandene Funktionalität erweitern kann. Dies wurde realisiert, indem bei der Erzeugung eines neuen „HistoryNMEASensor“ dem Konstruktor eine Referenz auf ein „NMEASensor“-Objekt übergeben wird. Dieses übernimmt daraufhin die Anbindung des Empfängers, während die Aufzeichnung der Historie vom umschließenden „HistoryNMEASensor“-Objekt durchgeführt wird. Für diesen Zweck stehen folgende

Methoden zur Verfügung: Mittels eines Aufrufs der Methode „startListening()“ wird ein Datensammlungslauf des umschlossenen „NMEASensor“-Objekts gestartet, dieser wird mittels eines Aufrufs der Methode „stopListening()“ wieder beendet.
Im Verlauf eines Datensammlungslaufes werden die bei jedem Zyklus erzeugten „GPSData“-Objekte, die ja sämtliche, vom GPS-Empfänger zurückgelieferte Daten enthalten, in die Felder eines Arrays vom Typ „GPSData“ geschrieben. Die Anzahl der „GPSData“-Objekte, die dieses Array aufnehmen kann, kann im Konstruktor durch Angabe eines dementsprechenden Wertes festgelegt werden, als Defaultwert wurde der Wert 20 gewählt. Dem Array neu hinzugefügte „GPSData“-Objekte werden solange in freie Felder des Arrays geschrieben, wie dies möglich ist. Sind alle Felder belegt, so werden neu diese auf dem Platz der jeweils am längsten eingetragenen „GPSData“- Objekte eingefügt, wobei die letzteren dabei überschrieben werden.
Zur Überprüfung, wieviele Objekte momentan im Array abgespeichert sind, steht die öffentliche Methode „getActEntries()“ zur Verfügung. Der Zugriff auf die abgespeicherten Objekte, z.B. zur nachfolgenden Auswertung, ist durch die beiden Methoden „getHistory“, die das gesamte Array und „getHistoryObject(int)“, die jeweils ein durch Angabe eines „int“-Wertes indiziertes „GPSData“-Objekt zurückliefert, möglich. Eine mögliche Verwendung dieser Klasse wird im diese Arbeit abschließenden Ausblick auf zukünftige Entwicklungen angegeben.

5.2.3 Das Package „Location“

Das Package „Location“ realisiert die Fähigkeit der Schnittstelle, Positionsdaten, die über ein Objekt des „GPS“-Packages erhalten wurden, darzustellen und weiterzuverarbeiten. So stehen u.a. Möglichkeiten für die Umwandlung geografischer Koordinaten von dem rechnerintern verwendeten „dd.mmmm“-Format in das gebräuchlichere „ddd°mm‘ss.ss‘“- Format für die geografische Länge und Breite bereit. Im folgenden werden die wichtigsten Zusammenhänge, sowie der Aufbau und der jeweilige Zweck der wichtigsten enthaltenen Klassen erklärt.

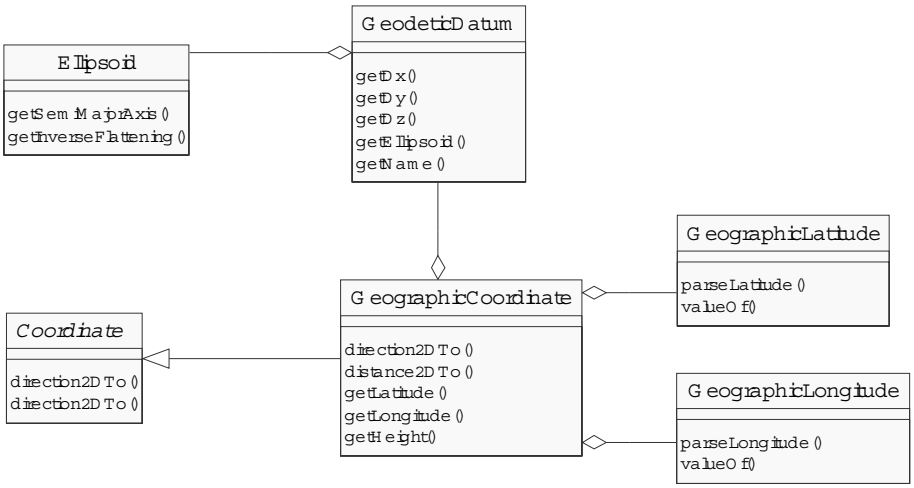


Abbildung 18: UML-Diagramm des "Location"-Packages

Die Klasse „Coordinate“

Diese Klasse dient als abstrakte Oberklasse für alle Klassen, die die Implementierung eines Positionierungsmodells realisieren. Es existiert eine große Anzahl von unterschiedlichen Positionierungsmodellen (siehe 3.1), deshalb wurde im Entwurf entschieden, für die entstandene Schnittstelle nur die Verwendung der geografischen Koordinatensysteme zu implementieren, die dafür ausschlaggebenden Gründe sind unter 4.2.3 nachlesbar. In der Klasse „Coordinate“ wurden einige Methoden als „abstrakt“ deklariert, unter anderem die Methoden „distanceTo(Coordinate)“ und „direction2DTo(Coordinate)“, deren Implementierung damit für alle erbenden Unterklassen obligatorisch ist. Die erste Methode dient der Abstandsbestimmung im dreidimensionalen Raum, die zweite Methode der Abstandsbestimmung in einem zweidimensionalen Raum, einer Näherung des genaueren dreidimensionalen Abstands, die für die Anforderungen vieler Anwendungen bereits ausreicht.

Die Klasse „Ellipsoid“

Hier werden die Referenzellipsoiden definiert, die zur Beschreibung eines geodätischen Datums benötigt werden (siehe nächster Abschnitt). Es werden außerdem noch Methoden zur Definition eigener Referenzellipsoiden bereitgestellt.

Die Klasse „GeodeticDatum“

Diese Klasse implementiert die jedem geografischen Modell zur Positionsangabe zugrunde liegenden geodätischen Daten. Wie bereits in 3.1 erläutert, wurden, um ein die tatsächlichen Form der Erde möglichst genau beschreibendes Erdmodell zu erhalten, Referenzellipsoiden definiert, die die tatsächliche Erdform möglichst genau beschreiben sollen. Zur Definition eines geodätischen Datums muß ein solcher Ellipsoid angegeben werden, außerdem können noch eventuelle Verschiebungsparameter definiert werden. Die Ellipsoiden, die hier verwendet werden, sind wiederum Instanzen der Klasse „Ellipsoid“. Es stehen zur Zeit 21 vordefinierte geodätische Daten zur Verfügung, diese beinhalten z.B. das wichtige WGS-84 Datum.

Die Klasse „GeographicCoordinate“

In dieser Klasse, die eine Unterklasse der bereits vorher angesprochenen, abstrakten Klasse „Coordinate“ ist, wird die Implementierung der geografischen Koordinaten zur Positionsangabe durchgeführt. Zur Erzeugung einer neuen geografischen Koordinate werden mehrere Konstruktoren bereitgestellt, durch deren Parameter sich das zugrunde liegende geodätische Datum, die geografische Länge, Breite und Höhe oder auch die jeweiligen, die Länge und Breite betreffenden Hemisphären festlegen lassen. Zusätzlich zur Angabe der Hemisphären mittels eigener Parameter wird zusätzlich die Festlegung dieser über das Vorzeichen von Länge und Breite unterstützt. Dabei wurde die in Europa gültige Konvention eingehalten, die besagt, daß positive Werte für die geografische Breite die nördliche Hemisphäre und positive Werte für die geografische Länge die östliche Hemisphäre bezeichnen. Entsprechend stellen negative Werte für die geografische Breite die südliche Hemisphäre und negative Werte für die geografische Länge die westliche Hemisphäre dar. Für die Angabe der geografischen Länge und Breite können einerseits Fließkommazahlen

verwendet werden, deren Wert sich gemäß dem Format „ddd.mmmm“ berechnet, anderseits besteht auch die Möglichkeit der Längen- und Breitenangabe gemäß dem gebräuchlicheren „ddd°mm’ss.ss‘“-Format. Hierfür wurden die beiden Klassen „GeographicLatitude“ und „GeographicLongitude“ entworfen und implementiert, deren Aufbau und Funktionalität wird weiter unten noch erläutert werden. Da diese Klasse, wie schon weiter oben erwähnt, eine Unterklasse der abstrakten Oberklasse „Coordinate“ ist, sind in dieser Klasse die Implementierungen der in der Oberklasse als „abstrakt“ deklarierten Methoden zu finden. Dabei wurde auf eine funktionelle Implementierung der Methoden „distanceTo(Coordinate)“ und „directionTo(Coordinate)“ bisher verzichtet, deren Funktionalität wird von den hier implementierten Methoden „distance2DTo(Coordinate)“, bzw. „direction2DTo(Coordinate)“ übernommen. Die den berechneten Distanzen und Richtungen zugrunde liegenden Formeln können u.a. in [WIL97] eingesehen werden.

Die Klassen „GeographicLatitude“ und „GeographicLongitude“

Diese beiden Klassen wurden aus Gründen der Steigerung der Verwendbarkeit der Schnittstelle entwickelt. Da die Verwendung des „ddd°mm’ss.ss‘“-Formats für die Angabe der geografischen Länge und Breite einer Position im allgemeinen am gebräuchlichsten ist, wurde, um den Benutzer vom Konvertieren zwischen den verschiedenen Formaten zu entlasten, eine Möglichkeit geschaffen, um Positionsangaben dieses Formats direkt verwenden zu können. Es stehen dazu Konstruktoren zur Verfügung, die eine geografische Länge oder Breite durch Angabe der vollen Grade, der vollen Winkelminuten und der als Fließkommazahl repräsentierten Winkelsekunden festlegen. Wahlweise kann noch durch Angabe einer Hemisphäre für die Länge und Breite diese festgelegt werden, andernfalls gilt die weiter oben beschriebene, in Europa gültige Konvention für negative Koordinatenangaben. Da die Position rechnerintern aus technischen Gründen dennoch als Fließkommazahl dargestellt wird, sind Methoden für die Konvertierung einer Länge/Breite in eine Fließkommazahl („getDoubleValue()“), bzw. zurück in das „ddd°mm’ss.ss‘“-Format („static parseLatitude(double)“, bzw. „static parseLongitude(double)“) vorhanden. Instanzen dieser Klassen werden von den Beispielsapplikationen immer dann verwendet, wenn eine Benutzereingabe einer Position erwartet wird. Auch hier sei auf die später folgenden Betrachtungen dieser Applikationen verwiesen.

5.3 Ablauf der Datensammlung

Im folgenden soll exemplarisch der Ablauf der Datensammlung von Positions– und Navigationsdaten, wie er in der entwickelten Kommunikationsbibliothek implementiert wurde, dargestellt werden.

Dazu soll im folgenden angenommen werden, daß der angeschlossene Empfänger als Instanz der Klasse „GenericNMEASensor“ eingebunden wurde, für diesen also keine Klasse zur Verfügung steht, die eventuell vom Empfänger unterstützte proprietäre Rahmen behandeln könnte. Als Folge werden in diesem Beispiel nur allgemeine NMEA-Rahmen zur Datensammlung beitragen. Die vom Empfänger erzeugten Rahmen eines Zyklus seien in dieser Reihenfolge: **$GPGGA, $GPGSV, $GPRMC**. Auch hier soll angemerkt werden, daß die Reihenfolge der Rahmen bei der kontinuierlichen Ausgabe dieser natürlich frei gewählt werden kann. Eine feste Reihenfolge wird erst durch die Definition eines geeigneten „Endrahmens“ eines Zyklus erzeugt. Die Gründe, die die Wahl des Rahmens „$GPRMC“ als Endrahmen eines Zyklus bedingten, sind bereits in Kapitel 5.2.2 besprochen worden.

Nachfolgend werden die einzelnen, vom Empfänger erzeugten Rahmen eines Rahmenzyklus, deren Datenfelder aus Anschauungsgründen mit exemplarischen Informationen gefüllt wurden, angegeben.

$GPGGA,142817,4844.6196,N,00905.8400,E,2,8,,,M,,M,,*50
$GPGSV,3,1,10,01,04,191,,02,10,285,36,07,14,319,,11,88,068,*78
$GPGSV,3,2,10,15,31,103,,16,13,241,42,21,09,061,,25,06,108,*7C
$GPGSV,3,3,10,29,25,046,,31,24,174,33,,,,,,,*70
$GPRMC,142817,A,4844.6196,N,00905.8400,E,000.0,000.0,230500,000.2,W*78

Der Ablauf der Datensammlung läßt sich nun in folgende Schritte aufgliedern:

1. Zu Beginn der Datensammlung durch Aufruf der Methode „startCommunication()“ wird ein Thread gestartet, der die vom Empfänger eintreffenden Datenrahmen zeilenweise von der angegebenen seriellen Schnittstelle liest und durch Überprüfung des ersten Datenfelds anhand des Rahmenbezeichners den Typ des jeweiligen Rahmens erkennt. Dieser Thread wird erst beendet, wenn die Methode „stopCommunication()“ aufgerufen wird.
2. Ist der Typ eines Rahmens bekannt, so wird zunächst in der die Anbindung realisierenden Klasse nach einer Methode zur Behandlung dieses Rahmentyps gesucht. Dies geschieht durch Aufruf der Methode „processSentence(String[])“. Ist keine derartige Methode vorhanden, wird der Rahmen an die Vaterklasse, also die Klasse „NMEASensor“ zur Behandlung weitergereicht. In diesem Fall werden alle Rahmen nach oben weitergereicht werden, denn die Klasse „GenericNMEASensor“ besitzt keine eigene Rahmenbehandlung.
3. Nachdem die zur Behandlung eines Rahmens passende Methode gefunden wurde, wird durch Aufruf dieser die im Rahmen enthaltenen Daten in den dafür vorgesehenen Variablen eines „GPSData“-Objekts abgelegt. Dieses Objekt ist nur als Arbeitsobjekt gedacht, dessen einziger Zweck das Sammeln von Daten während eines Rahmenzyklus ist.
4. Wird der vorher definierte Endrahmen eines Zyklus erkannt, so werden durch Aufruf der entsprechenden Methode noch die in diesem übermittelten Informationen in das oben erwähnte GPSData-Objekt übertragen. Danach wird durch Aufruf der Methode „update()“ der Datensammlungsprozeß für diesen Zyklus abgeschlossen.
5. Durch Aufruf der Methode „update()“ in der jeweiligen, die Anbindung des Empfängers realisierenden Klasse wird zuerst die „update()“-Methode der Vaterklasse „NMEASensor“ aufgerufen. In dieser werden die Variablen gesetzt, die die Rückgabewerte für die Methoden für den Direktzugriff auf allgemeine wichtige Daten darstellen. Nach Abarbeitung dieser Methode werden in der aufrufenden „update()“-Methode die Variablen gesetzt, die die Rückgabewerte für die Methoden zum Direktzugriff auf herstellerspezifische wichtige Informationen bereitstellen. In diesem Beispiel wird der zweite Schritt entfallen, denn die Klasse „GenericNMEASensor“ bietet gegenüber ihrer Vaterklasse keine zusätzlichen Methoden zum Direktzugriff auf Daten an. Im Verlauf der Abarbeitung der „update()“-Methode werden alle im „GPSData“-Arbeitsobjekt angesammelten Daten in ein „GPSData“-Objekt übertragen, auf das von externen Anwendungen mittels der Methode „getCurrentGPSData()“ zugreifbar ist. Zuletzt wird das Ende dieses Zyklus und das Bereitstehen neuer Informationen in oben erwähntem „GPSData“-Objekt durch das Setzen einer entsprechenden Variable mittels der Methode „setUpdated(boolean)“ nach außen bekanntgegeben. Anwendungen können nun bis zum Ende des folgenden Rahmenzyklus auf die gesammelten Daten zugreifen.

Durch den Erhalt obiger Rahmen vom Empfänger würden z.B. folgende Daten nachfolgend zur Verfügung stehen (Angabe jeweils mit Bezeichnern der Rahmen über die diese erhalten wurden):

- UTC-Zeit der Positionsbestimmung: 14:28:17 ($GPGGA, $GPRMC),
- Geografische Breite der Position:48° 44.6196‘ Nord ($GPGGA, $GPRMC)
- Geografische Länge der Position: 9° 5.8400‘ Ost ($GPGGA, $GPRMC)
- Qualität der durchgeführten Positionsbestimmung: DGPS ($GPGGA)
- Anzahl der Satelliten, mittels derer die Position bestimmt wurde: 8 ($GPGGA)
- Anzahl der Satelliten, die „sichtbar“ waren: 10 ($GPGSV), die anderen in diesem Rahmen übertragenen Daten geben für jeden dieser 10 Satelliten die PRN (***P**seudo **R**andom **N**oise*, die eindeutige Identifikation eines Satelliten, auch als „Pseudo Random Code“ bekannt), den Azimut dieses Satelliten, dessen Erhebung und Signalstärke an.

5.4 Mögliches Vorgehen bei einer Erweiterung der Schnittstelle

Folgend wird eine kurze Anleitung für eine eventuelle Erweiterung der Kommunikationsbibliothek um eine weitere Klasse angeben, die die Anbindung eines neuen Empfängertyps realisieren soll.

1. Identifikation der allgemeinen und proprietären NMEA 0183 Rahmen, deren Behandlung bisher nicht implementiert wurde. Methoden zur Behandlung allgemeiner NMEA-Rahmen sollten in der Basisklasse „NMEASensor“, Methoden zur Behandlung proprietärer Rahmen direkt in die neu erstellte Klasse eingefügt werden. Sollten proprietäre Rahmen verwendet werden, so sollte in den Konstruktoren die von der Vaterklasse geerbte Variable „basicNMEA“ auf den Wert *false* gesetzt werden.
2. Identifikation eines Endrahmens, dessen Erhalt das Ende eines Zyklus anzeigt.
3. Erweiterung der Klasse „GPSData“ um eventuell neu hinzugekommene Datenfelder. Dabei muß bei der Erweiterung auch eine Einordnung der neu hinzugekommenen Datenfelder in eine der beiden Klassen „allgemeine Variablen“ und „proprietäre Variablen“ erfolgen, was durch Modifikation der Methode „resetAllValues(boolean)“ geschehen kann.
4. Überschreiben der „processSentence(String[])“-Methode der Vaterklasse. In dieser Methode sollte die Zuordnung eines von der seriellen Schnittstelle gelesenen Rahmens zu der für dessen Behandlung vorgesehenen Methode erfolgen. Unter anderem sollte in dieser Methode der Aufruf der „update“-Methode nach Erkennens des zuvor definierten Endrahmens eines Zyklus erfolgen.
5. Überschreiben der „update“-Methode der Vaterklasse in der neu erzeugten Klasse. Hier können z.B. Variablen belegt werden, die die Rückgabewerte für neu implementierte Methoden zum Direktzugriff auf wichtige Daten beinhalten. Außerdem muß hier durch Aufruf der Methode „setUpdated(boolean)“ mit dem Parameter „*true*“ bekannt gemacht werden, daß ein Update durchgeführt wurde und nun aktuelle Daten zur Verfügung stehen.
6. Implementierung der in der Vaterklasse als „abstrakt“ deklarierten Methoden (z.B. startCommunication()).

6 Die Anwendungen „GPSApplication“ und „MapTool“

Zusätzlich zur Hauptaufgabe, der Entwicklung einer Bibliothek für die Kommunikation mit einem GPS-Empfänger, sollte noch eine die entwickelte Bibliothek verwendende Beispielsanwendung entworfen und implementiert werden, mit deren Hilfe die wichtigsten, von einem GPS-Empfänger zu erhaltenden Daten angezeigt und zum Teil auch aufgezeichnet werden könnten. Weiterhin sollte ein Tool entwickelt werden, mit dessen Hilfe die eigene Position auf einer grafischen Karte einzeichenbar sein sollte. Diese beiden Anwendungen würden einen Benutzer in die Lage versetzen, sein GPS-Gerät auch ohne Implementierung eigener Routinen testen zu können. Im folgenden soll die Entwicklung der entstandenen Anwendungen, sowie deren Funktionalität beschrieben werden.

6.1 Die Anwendung „GPSApplication“

6.1.1 Vorgaben

Entwickelt werden sollte eine die Bibliothek verwendende, kommandozeilenorientierte Umgebung, die die Position, die Kompaßrichtung, die Geschwindigkeit, die aktuelle Qualität der Positionsbestimmung und die Distanz zu einem bestimmbaren Punkt anzeigen kann. Zusätzlich sollte noch die Möglichkeit existieren, eine zurückgelegte Strecke aufzuzeichnen.

6.1.2 Entwurfsentscheidungen

Die relativ abstrakt gestellte Forderung nach Anzeige der aktuellen Qualität der Positionsbestimmung wurde durch Anzeige der drei Präzisionsminderungsmaße PDOP, HDOP und VDOP (siehe 3.2.2), sowie durch Anzeige des Typs der Positionsbestimmung (zwei- oder dreidimensional) und der Art des hierfür verwendeten Verfahrens (GPS, bzw. DGPS) gelöst. Statt den ursprünglich geplanten kommandozeilenorientierten Tools wurde beschlossen, diese in einer grafischen Benutzeroberfläche zu vereinen. Weitere, die Funktionalität der Anwendung über die gestellten Vorgaben hinaus erweiternde Entwurfsentscheidungen, waren unter anderem:

- Der Umfang der darzustellenden GPS-Daten wurde erweitert, um Anzeigen für das aktuelle Datum und die Tageszeit, an welcher die aktuelle Position bestimmt wurde, die Höhe, in der sich der Empfänger zum Zeitpunkt der Messung befand und die Anzahl der „sichtbaren“ Satelliten und die Anzahl der Satelliten, mittels derer schließlich die aktuelle Position bestimmt wurde. Dabei wurden alle Anzeigen als selektierbar entworfen, d.h. es sollte möglich sein, diese zu aktivieren und zu deaktivieren.
- Die mögliche Auswahl eines Empfängertyps. Es sollte einem Benutzer möglich sein, zwischen der Verwendung des angeschlossenen GPS-Empfänger als einer der beiden in der Bibliothek implementierten Typen „Garmin25LP“ oder „GenericNMEA“ zu wählen. Zusätzlich wurde noch die Implementierung eines „Simulationsmodus“ beschlossen, mit Hilfe dessen auch ohne Anschluß eines realen Empfängers die Funktionalität der Anwendung ersichtlich sein würde.
- Die Auswahl der seriellen Schnittstelle, über die ein Empfänger angeschlossen ist.
- Die Aufzeichnung der vom Empfänger über die Schnittstelle gesendeter Datenrahmen. Der Entwurf sah weiterhin vor, daß die dabei erzeugten Dateien als Eingabe für den oben angesprochenen Simulationsmodus verwendbar sein sollten.

- Um die Anwendung an die Bedürfnisse der verschiedenen späteren Benutzer anpassen zu können, wurde im Entwurf eine Speichermöglichkeit der vom Benutzer vorgenommen Einstellungen vorgesehen. Bei den Einstellungen handelt es sich dabei vor allem um die verschiedenen selektierbaren Datenanzeigen, deren aktuelle Einstellungen abspeicherbar und bei einer folgenden Ausführung der Anwendung wiederherzustellen sein sollten.

6.1.3 Beschreibung der Funktionalität

Die Implementierung dieser Vorgaben und weiterführenden Entscheidungen wurde in einem zusätzlichen Package, dem „GUI“-Package, gekapselt. Der Aufbau dieses Packages soll hier nicht im einzelnen dargestellt werden.

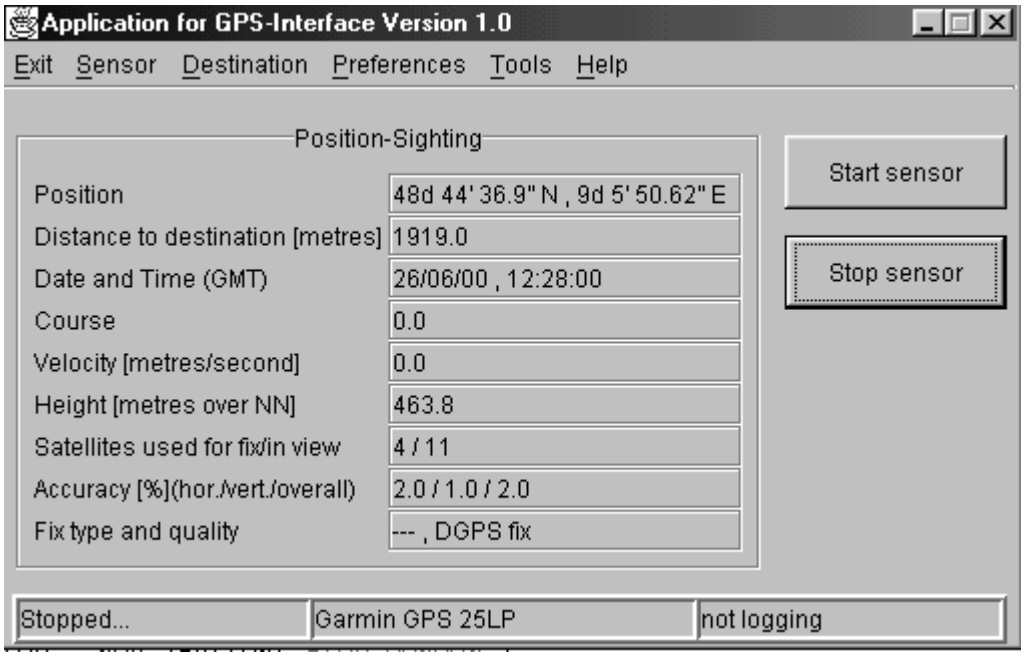


Abbildung 19: Grafische Benutzeroberfläche der Anwendung GPSApplication

Die Oberfläche wurde, wie aus der obigen Abbildung ersichtlich ist, bewußt einfach gehalten. Alle Funktionen sind über die am oberen Rand befindlichen Menüleiste auswählbar. Um die Auswahl der wichtigsten Funktionen zu vereinfachen, wurden zum Starten und Stoppen eines Datensammlungslaufes zwei Schaltflächen eingefügt. Links dieser Schaltflächen befindet sich das die unterschiedlichen GPS-Daten anzeigende Display. Am unteren Rand der Oberfläche ist zur Anzeige von Statusinformationen ein dreigeteilter Statusbalken eingefügt worden. Folgend werden die eben genannten Bestandteile der Oberfläche detaillierter erklärt.

Die Menüleiste

Die Menüleiste enthält insgesamt 6 Menüs, wobei diese wiederum teilweise in Untermenüs gegliedert sind. Die Funktion des ersten Menüs („`Exit`“), das nur einen gleichnamigen Menüpunkt enthält, ist das Schließen der Anwendung und aller von ihr erzeugten Fenster. Das zweite Menü („`Sensor`“) enthält sämtliche Funktionen, die für die Wahl der Anschlußparameter eines GPS-Empfängers nötig sind. Im einzelnen sind dies die Auswahl einer seriellen Schnittstelle, über die der Empfänger angeschlossen ist und die Festlegung des

Typs des angeschlossenen Empfängers. Zur Wahl stehen hier derzeit 3 Alternativen, im einzelnen sind dies der Typ „`Garmin 25LP`“, der allgemeine Typ „`Generic NMEA`“ oder die Wahl des bereits angesprochenen Simulationsmodus. Bei Auswahl des letzteren wird die Angabe der Position einer Datei gefordert, in der die Simulationsdaten gespeichert sind. Im Menü „`Sensor`“ läßt sich ebenfalls der Start bzw. das Stoppen eines Datensammlungslaufes durch Wahl des entsprechenden Menüpunkts veranlassen. Das dritte Menü („`Destination`“) dient der Angabe eines Zielpunkts, wobei daraufhin die berechnete Entfernung der aktuellen Position zu diesem Punkt im entsprechenden Feld des Displays angezeigt wird. Das Dialogfenster, daß nach Auswahl dieses Menüpunktes zur Eingabe einer Zielposition erscheint, ist in Abbildung 20 dargestellt. Für die Eingabe einer Zielposition gelten dabei folgende Regeln:

- Die geografische Breite muß gemäß dem üblichen „dd° mm‘ ss.ss‘“-Format mit zusätzlicher Angabe einer Hemisphäre eingegeben werden. Dabei sind folgende Wertebereiche gültig: 0 < dd < 90, 0 < mm < 60, 0.0 < ss.ss < 60.00, N oder S für die Hemisphäre.
- Die geografische Länge muß gemäß dem üblichen „ddd° mm‘ ss.ss‘“-Format mit zusätzlicher Angabe einer gültigen Hemisphäre eingegeben werden. Dabei sind folgende Wertebereiche gültig: 0 < ddd < 180, 0 < mm < 60, 0.0 < ss.ss < 60.00, E oder W für die Hemisphäre.

Eine Eingabe wird nach Betätigen der „`OK`“- Schaltfläche auf Gültigkeit überprüft. Fällt diese Überprüfung negativ aus, so wird die Eingabe nicht akzeptiert, der Benutzer kann daraufhin gegebenenfalls Änderungen an den bisher eingegebenen Werten vornehmen. Führt die Überprüfung zu einem positiven Ergebnis, so wird die Eingabemaske geschlossen und die neue Zielposition der Anwendung zur Verfügung gestellt.

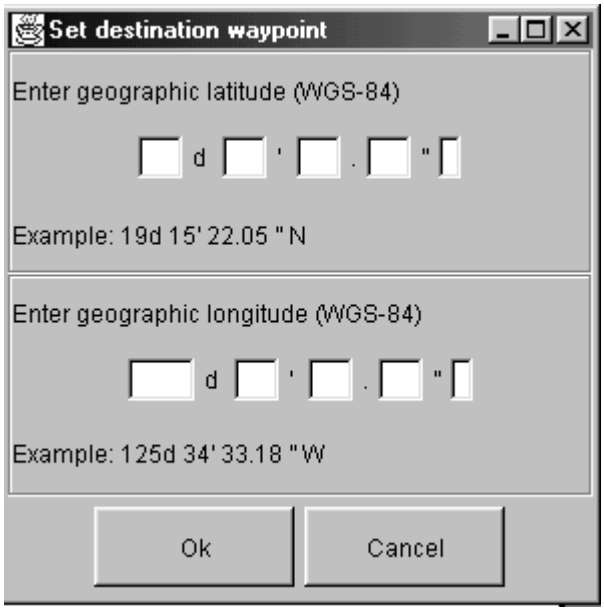


Abbildung 20: Dialog für die Eingabe einer Zielposition

Das folgende Menü, „`Preferences`“, dient erstens dem Aktivieren bzw. Deaktivieren der Anzeigen für die verschiedenen GPS-Daten. Dies läßt sich nach Auswahl des Untermenüs „`Position-Sighting Options`“ durch Auswählen der entsprechenden Menüpunkte durchführen. Der zweite Zweck des „`Preferences`“-Menüs ist die Festlegung der

Aufzeichnungsfrequenz für die beiden möglichen Aufzeichnungsarten. Hier handelt es sich zum einen um die Aufzeichnung von Positionsdaten („`Position Logging`“), mit deren Hilfe ein zurückgelegter Weg nachvollzogen werden kann, und zum anderen um die Aufzeichnung der vom Empfänger erzeugten NMEA-Rahmen, die für die Simulation eines GPS-Empfängers verwendet werden kann. Wählbare Aufzeichnungsfrequenzen sind in beiden Fällen :

- Nie (auf die Aufzeichnung wird verzichtet),
- jeder Rahmenzyklus (jede neue Position, bzw. alle erhaltenen Rahmen werden aufgezeichnet),
- jeder zehnte Rahmenzyklus,
- jeder hundertste Rahmenzyklus.

Zusätzlich lassen sich noch die Dateien, in denen die Aufzeichnungen jeweils gespeichert werden sollen, durch Wahl der Menüpunkte „`Set Position Logfile`“, bzw. „`Set Sentences Logfile`“ spezifizieren. Alle Einstellungen, die im „`Preferences`“-Menü getätigt wurden, können durch Auswahl des Menüpunkts „`Save as Default`“ auf nicht flüchtigem Speicher gesichert werden, von wo sie bei einer nachfolgenden Ausführung der Anwendung wieder eingelesen werden.
Das vorletzte Menü („`Tools`“) dient dem Aufruf der weiter unten beschriebenen Anwendung „`MapTool`“, mit der auf einer grafischen Karte der aktuelle Aufenthaltsort eingezeichnet werden kann.
Im letzten, mit „`Help`“ bezeichneten Menü kann durch die Auswahl des Menüpunktes „`About`“ nähere Angaben über Autor usw. erhalten werden. Durch Auswahl des zweiten Menüpunktes „`Help`“ wird eine kurze Onlinehilfe über die Anwendung angezeigt.

Das Datendisplay

Das Datendisplay bietet Anzeigen für folgende GPS-Daten an:

- Die aktuelle Position (geografische Koordinaten),
- die Distanz zu einer wählbaren Position,
- UTC-Zeit und UTC-Datum,
- der verfolgte Kompaßkurs,
- die aktuelle Geschwindigkeit (Meter/Sekunde),
- die aktuelle Höhe,
- die Anzahl der „sichtbaren“ und Anzahl der bei der Positionsbestimmung beteiligten Satelliten,
- eine Genauigkeitsabschätzung der Positionsbestimmung durch Angabe der dafür vorgesehenen Maße PDOP, HDOP, VDOP,
- der Typ (zwei- oder dreidimensional) und Art (GPS oder DGPS) der Positionsbestimmung.

Der Statusbalken

Der am unteren Rand der grafischen Oberfläche befindliche Statusbalken ist in drei Statusfelder aufgeteilt. Im linken Statusfeld ist der aktuelle Status eines Datensammlungslaufes dargestellt. Hier mögliche Einträge sind z.B. „not running“, dies ist der Fall, wenn noch kein Lauf gestartet wurde, „looking for sensor“, hier wird überprüft, ob ein Empfänger, der mit dem gewählten Typ kompatibel ist, an der spezifizierten seriellen Schnittstelle angeschlossen ist, „computing valid position fix“, zur Anzeige, daß die momentan erhältlichen Positionsdaten ungültig sind, oder auch „running“, zur Anzeige, daß der aktuelle Datensammlungslauf gültige Positionsdaten liefert.
Das mittlere Statusfeld stellt eine Anzeige dar, die den Typ des gewählten Sensors angibt. Zur Zeit mögliche Werte sind hier „Generic NMEA Sensor“, „Garmin GPS 25 LP“ und „Simulating from file“.
Im letzten Statusfeld werden schließlich die verschiedenen gewählten Aufzeichnungsmodi angezeigt. Hier mögliche Werte sind „Sentence Logging enabled“, was die Aufzeichnung der vom Empfänger gesendeten Rahmen anzeigt, „Position Logging enabled“, was die Aufzeichnung von Positionsdaten, und „Sentence and Position Logging enabled“, das die Wahl beider Aufzeichnungsarten signalisiert. Sollte kein Aufzeichnungsmodus aktiviert worden sein, so wird dies durch die Anzeige „not logging“ widergespiegelt.

6.2 Die Anwendung „MapTool“

6.2.1 Vorgaben

Die Vorgaben, die für die Erstellung dieser Anwendung gestellt wurden, waren sehr abstrakt gehalten. Es wurde lediglich gefordert, daß die zu entwickelnde Anwendung es ermöglichen sollte, in einer grafische Karte den momentanen Aufenthaltspunkt anzuzeigen. Weitere Vorgaben wurden nicht gestellt.

6.2.2 Entwurfsentscheidungen

- Es sollte die Möglichkeit bestehen, mit Hilfe der Anwendung eigene Karten zu definieren und diese danach abspeichern zu können. Gleichfalls sollten bereits erstellte und gesicherte Karten wieder durch die Anwendung zu laden sein.
- Eine bereits definierte Karte sollte redefinierbar sein, d.h. es sollte die Möglichkeit angeboten werden, die Attribute einer Karte wieder zu verändern. Unter Attributen einer Karte wird hier die Angabe der geografischen Koordinaten der linken unteren Ecke und der geografischen Koordinaten der rechten oberen Ecke verstanden.
- Die Anwendung sollte die momentane Position möglichst selbsttätig aktualisieren.
- Da die Möglichkeit besteht, daß eine gewählte Karte größer ist als die angebotene Anzeigefläche, d.h. zu jedem Zeitpunkt nur ein Ausschnitt aus dieser Karte angezeigt werden kann, sollte es einem Benutzer möglich sein, den Kartenausschnitt, der die aktuelle Position enthält, schnell zur Darstellung auszuwählen.

6.2.3 Beschreibung der Funktionalität

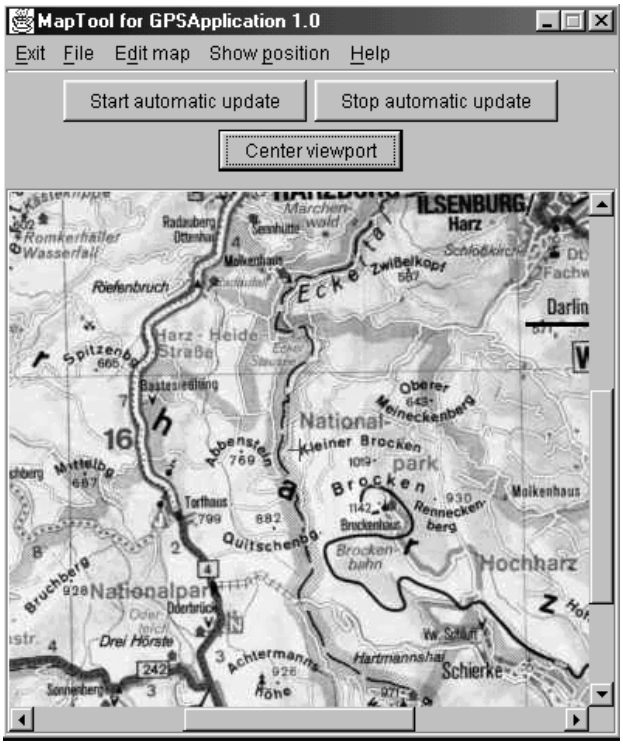


Abbildung 21: Die Anwendung "MapTool"

Die aus den Entwurfsentscheidungen hervorgehende Anwendung „MapTool“, genauer deren grafische Benutzerschnittstelle, gliedert sich in folgende Komponenten:

Die Menüleiste

Mittels der Menüleiste können sämtliche, von der Anwendung bereitgestellte Funktionen ausgewählt werden. Mittels des zweiten Menüs, „`File`“, kann durch Auswahl des Menüpunkts „`Load Map`“ eine bereits erstellte Karte durch Angabe ihrer Position im Verzeichnisbaum geladen und zur Anzeige im Darstellungsfenster gebracht werden. Eine neue Karte kann durch Wahl des Menüpunkts „`Load raw image`“ erstellt werden. Dabei wird zunächst durch Wahl einer Grafik (*JPG*, *GIF*) das der zu erzeugenden Karte zugrunde liegende grafische Objekt eingelesen. Anschließend werden nacheinander zwei Dialoge dargestellt, wobei der Erste der Eingabe der geografischen Koordinaten des linken, unteren Ecke der zu definierenden Karte, der Zweite der Eingabe der rechten, oberen Ecke dient. Dabei gelten die gleichen Konventionen für die gültigen Wertebereiche wie bei der unter 6.1.3 beschriebenen Eingabe einer Zielposition. Durch Auswahl des Menüpunkts „`Save Map`“ kann eine erstellte grafische Karte abgespeichert werden.
Das dritte Menü, „`Edit Map`“, ermöglicht durch Auswahl des einzigen Menüpunktes „`Specify Corners`“ die Redefinition einer bereits erstellten Karte. So ist es hier möglich, die geografischen Koordinaten der beiden, eine Karte definierenden Ecken neu festzulegen.
Im vierten Menü, „`Show position`“ wird Funktionalität des Einzeichnens der aktuellen Position in eine grafische Karte angeboten. Voraussetzung dazu ist eine im Zustand „running“ befindliche Instanz der Anwendung „GPSApplication“. Durch Wahl des Menüpunktes „`Start drawing current position`“ wird die jeweils bestimmte Position in die gewählte

Karte eingezeichnet, wenn sich die ermittelte Position auf der aktuellen Karte darstellen läßt. Sollte dies der Fall sein, so wird an der berechneten Stelle der Karte ein rotes Kreuz sichtbar. Das Einzeichnen der aktuellen Position kann entweder durch Wahl des Menüpunktes „`Stop drawing current position`“, oder durch Stoppen des Datensammlungslaufes in der unterliegenden Instanz der Anwendung „GPSApplication“ erfolgen. Durch Auswahl des Menüpunktes „`Center map to position`“ wird der angezeigte Kartenausschnitt auf die momentane Position zentriert.

Direktanwahl wichtiger Funktionen durch Schaltflächen

Wie schon bei der Entwicklung der Anwendung „GPSApplication“ wurde auch bei der Entwicklung der Anwendung „MapTool“ beschlossen, die Auswahl der wichtigsten Funktionen nicht nur über die Menüleiste, sondern auch direkt über entsprechende Schaltflächen möglich zu machen. Für diesen Zweck wurden der grafischen Benutzerschnittstelle drei Schaltflächen hinzugefügt, mittels derer die Funktionen *Aktivierung* bzw. *Deaktivierung* des periodischen Einzeichnens der aktuellen Position und *Karte auf aktuelle Position zentrieren* direkt auswählbar sind.

6.3 Ein aufgetretenes Problem

Zur Implementierung der beiden Anwendungen wurde die für die Entwicklung grafischer Benutzerschnittstellen entwickelten *Swing*-Packages verwendet. Diese Packages sind im Kern der Java 2-Plattform enthalten und wurden von der Firma Sun entworfen, um in den folgenden Jahren die bisherig für die Entwicklung von grafischen Benutzerschnittstellen zur Verfügung stehenden AWT-Klassen (***A**bstract **W**indowing **T**oolkit)* zu ersetzen. Vorteile der *Swing*-Architektur gegenüber der Architektur der AWT-Klassen sind unter anderem der Verzicht auf nativen Code bei der Implementierung der *Swing*-Klassen. Weitere Informationen über die *Swing*-Technologie sind bei http://www.java.sun.com nachlesbar. Allerdings ist die Entwicklung der *Swing*-Packages noch nicht voll ausgereift, so kann z.B. bei Verwendung der entwickelten Anwendungen „GPSApplication“ und „MapTool“ nachfolgend geschilderter Laufzeitfehler auftreten. Dieser ist als offizieller Java-Bug (Bug-ID: 4200433) bekannt. Das Auftreten dieses Laufzeitfehlers äußert sich in einem ungültigen Seitenzugriff im Modul „AWT.dll“ und dem darauffolgenden Abbruch der Anwendung. Laut den Bugreports der JDC (***J**ava **D**evelopers **C**onnection)* kann dieser Fehler bei häufig auftretenden Zugriffen auf *Swing*-Komponenten wie „JTextFields“ oder „JLabels“ bei gleichzeitiger Verwendung mehrerer Threads auftreten. Da dieser Fehler noch nicht seitens der Firma Sun behoben wurde, wurde versucht, die Zugriffe auf Textkomponenten, wie z.B. den einzelnen Komponenten des Datendisplays der Anwendung „GPSApplication“, zu minimieren. Dazu werden z.B. die Felder des Datendisplays erst dann aktualisiert, wenn tatsächlich ein neuer Wert zur Verfügung steht. Aus den Bugreports ist ebenfalls ersichtlich, daß häufige Benutzerinteraktionen die Häufigkeit des Auftretens dieses Fehlers noch zu steigern vermögen. Mögliche Maßnahmen, die ein Benutzer zur weitgehenden Vermeidung dieses Fehlers ergreifen kann, könnten dann z.B. die Deaktivierung derjenigen Komponenten des Datendisplays, deren Informationen für den jeweiligen Anwendungszweck unnötig sind, oder die Beschränkung von Benutzerinteraktionen auf unkritische Zeitpunkte, wie z.B. die „offline“ erfolgende Spezifikation einer Zielposition sein.

7 Durchgeführte Tests

In diesem Kapitel soll auf durchgeführte Tests eingegangen werden. Dazu sollen beispielhaft Daten, die bei einem Testlauf auf dem Weg vom Rechenzentrum der Universität Stuttgart über die Fraunhofer-Gesellschaft zum Studentenwohnheim „Bauhäusle“ gewonnenen wurden, besprochen werden. Dazu soll anhand einer Karte, in der die verfolgte Route eingezeichnet wurde, die Korrelation zwischen den vom GPS-Empfänger gelieferten Daten und den „realen“ Positions- und Navigationsdaten aufgezeigt werden. Abschließend erfolgt eine Bewertung der Übereinstimmung dieser beiden Wertegruppen.

7.1 Die verfolgte Route

Abbildung 22: Karte der Universität Stuttgart (Vaihingen) mit begangener Strecke

Zusätzlich zu den weiter unten dargestellten Daten wurde die Position der eingezeichneten Wegpunkte bestimmt. Die Angaben sind als WGS-84 Koordinaten aufzufassen.

- **Start** (Rechenzentrum der Universität Stuttgart) : 48° 44‘ 25.34‘‘ Nord, 9° 5‘ 42.83‘‘ Ost (Beginn der Aufzeichnung).
- **FHG** (Fraunhofer-Gesellschaft: 48° 44‘ 25.09‘‘ Nord, 9° 5° 56.59‘‘ Ost (8 Minuten nach Aufzeichnungsbeginn).
- **Kreuzung 1**: 48° 44‘ 26.84‘‘ Nord, 9° 6‘ 12.91‘‘ Ost (12 Minuten nach Aufzeichnungsbeginn).
- **Kreuzung 2**: 48° 44‘ 35.05‘‘ Nord, 9° 6‘ 10.07‘‘ Ost (16 Minuten nach Aufzeichnungsbeginn).
- **Brücke**: 48° 44‘ 40.81‘‘ Nord, 9° 6‘ 6.85‘‘ Ost (19 Minuten nach Aufzeichnungsbeginn)
- **Ziel** (Studentenwohnheim): 48° 44‘ 36.80‘‘ Nord, 9° 5‘ 52.10‘‘ Ost (24 Minuten nach Aufzeichnungsbeginn → Ende der Aufzeichnung).

7.2 Auswertung der gesammelten Informationen

Durch Aufzeichnung wurden folgende Datenarten für eine spätere Auswertung gesammelt:

- Die Anzahl der „sichtbaren“ Satelliten,
- die Anzahl der Satelliten, die jeweils für die Bestimmung der Position benutzt wurden,
- die PDOP-Metrik zur Abschätzung der Genauigkeit der Positionsbestimmung,
- die Art der Positionsbestimmung (GPS oder DGPS, wenn Korrektursignale zu empfangen waren).

Die durch die Messungen der ersten drei Datentypen bestimmten Werte sind im folgenden in Diagrammform dargestellt. Als Intervall für die Messung diskreter Werte wurde eine Minute festgelegt.
Zusätzlich bestimmte Daten, die hier nicht weiter dargestellt werden sollen, waren z.B. die Position, UTC-Zeit und der Kurs, der jeweils verfolgt wurde. Zum Umfang der erstellten Kommunikationsbibliothek gehört die Aufzeichnung der NMEA-Rahmen, die vom Empfänger gesendet wurden. Mit Hilfe dieser Datei, „Uni_Vaihingen.clog“ kann mittels des Simulationsmodus der Anwendung „GPSApplication“ die verfolgte Strecke rekonstruiert werden.

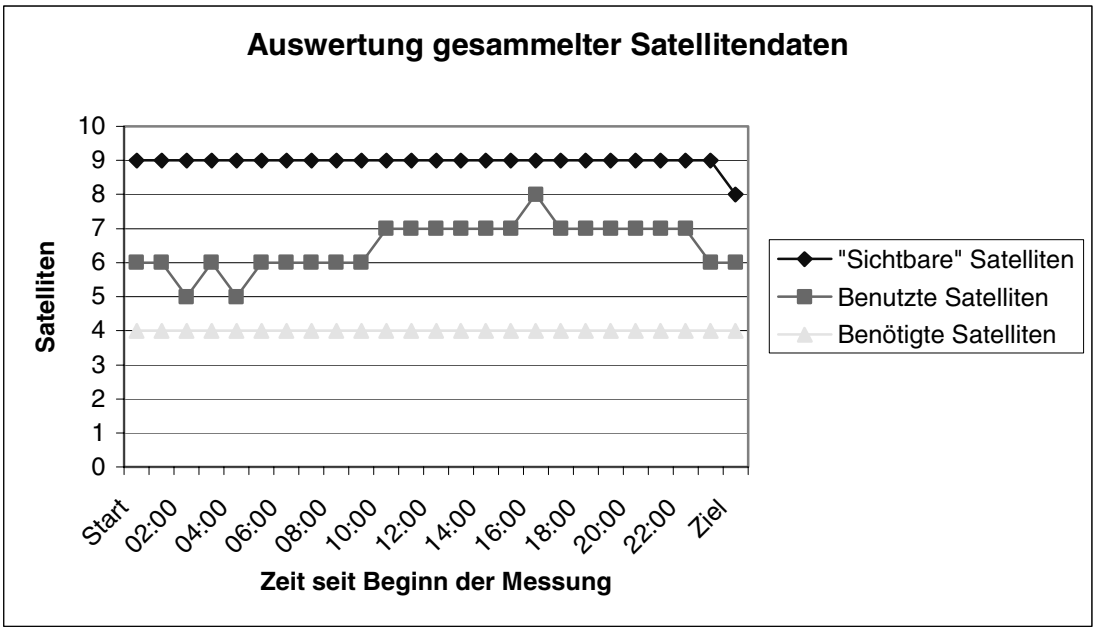


Abbildung 23: Diagramm der gesammelten Satellitendaten

Auffällig ist hier vor allem, daß keine, oder nur geringe Abschattungseffekte durch die mehrstöckigen Gebäude des Studentendorfes „Allmandring“ beim Durchqueren dieses Gebietes (Aufzeichnungsminuten 20 bis 24) auftraten. Dies zeigt sich an der nahezu konstanten Anzahl sichtbarer Satelliten. Die allgemein relativ niedrige Anzahl der „sichtbaren“ Satelliten (maximal 12) wurde wohl durch das am Aufzeichnungstag vorherrschende Wetter (stark bewölkt, teilweise leichter Nieselregen) bedingt, das keine optimalen Bedingungen bot.

Das zweite Diagramm (Abbildung 24) stellt die Entwicklung des PDOP-Werts (siehe 3.2.2) im zeitlichen Verlauf der Aufzeichnung dar. Auffällig ist hier die merkbare Verschlechterung dieses Wertes, die sich bei Durchquerung des Studentendorfes ergab. Dies kann durch ein Ansteigen des *Multipath*-Fehlerpotentials (siehe 2.3.4) erklärt werden.

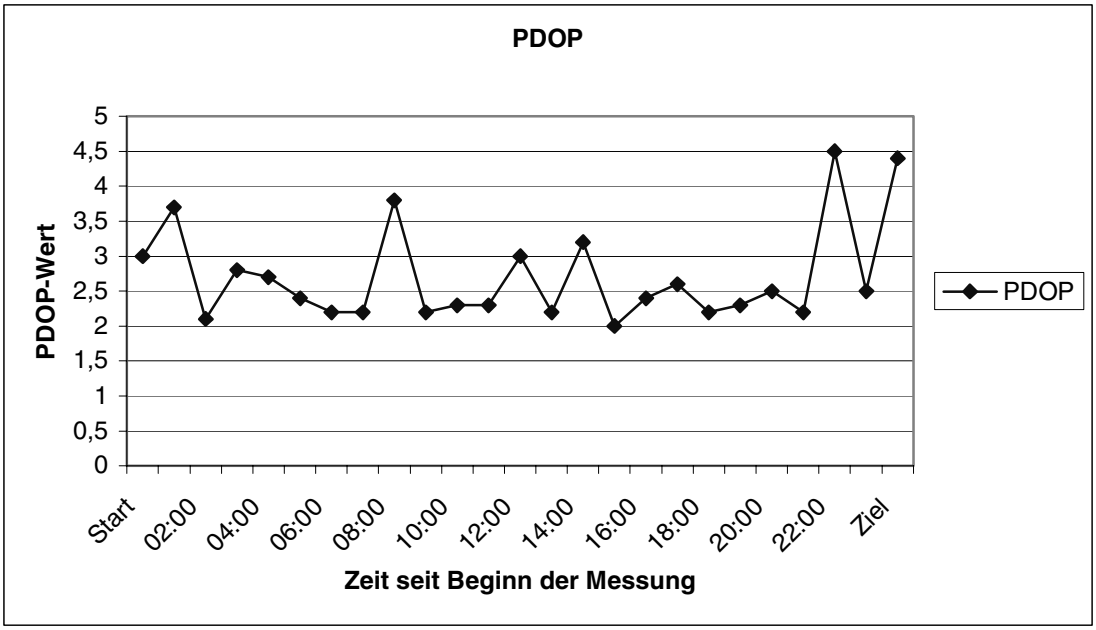


Abbildung 24: Diagramm der gesammelten PDOP-Werte

Generell kann aber festgestellt werden, daß die erzielte Positionsgenauigkeit für die äußeren Umstände (Wetter, Gebäude) durchaus zufriedenstellend war. Ein durchschnittlicher PDOP-Wert von 3 wird allgemein als „gute“ Meßgenauigkeit angesehen.

Abschließend soll noch kurz auf die Verfügbarkeit eines DGPS-Korrektursignals eingegangen werden. Ein solches konnte während der gesamten Aufzeichnungszeit empfangen werden, unabhängig von den äußeren Umständen, die jeweils auftraten. Dies ist eine mögliche Erklärung für den durchgängig zufriedenstellenden PDOP-Wert, der ja auf eine genaue Positionsbestimmung schließen läßt.

8 Ausblick

Zum Abschluß dieser Ausarbeitung soll noch auf mögliche Erweiterungsmöglichkeiten und Verwendungszwecke der erstellten Kommunikationsbibliothek eingegangen werden. Mögliche, sinnvolle Erweiterungen der bisher angebotenen Funktionalität können z.B. sein:

- Erweiterung des „Location"-Packages um Klassen, die eine Verwendung auch anderer Positionierungsmodelle neben den bereits implementierten geografischen Modellen ermöglichen. Dabei ist an erster Stelle die Klasse der konformen Koordinatensysteme zu nennen, die als *Gauß-Krüger*-Koordinatensysteme speziell in Deutschland häufig verwendet werden.
- Implementierung neuer Klassen, die es ermöglichen, die durch proprietäre Rahmen anderer Hersteller als Garmin erhältlichen Daten auszunutzen, d.h. Erweiterung des „GPS"-Packages. Ein weiterer bekannter Hersteller von GPS-Geräten ist z.B. die Firma Magellan.
- Implementierung neuer Methoden in den bisher vorhandenen Klassen, die die Funktionalität dieser zu erweitern vermögen. Denkbar sind in diesem Zusammenhang vor allem neue Methoden für den Direktzugriff auf noch zu identifizierende, logisch zusammengehörige Datentypen, beispielsweise für den direkten Zugriff auf die 4 vom Empfänger angebotenen **DOP*-Metriken (*PDOP*, *HDOP*, *VDOP* und *TDOP*) oder Methoden für die Auswertung bisher nicht berücksichtigter NMEA 0183 Rahmen. Dies würde dann auch zu Erweiterungen in der „GPSData"-Klasse führen, in der die neu zu erlangenden Daten abgespeichert werden (eine neue Variable, die den Wert speichert und zwei Methoden zum Setzen und Abfragen des Werts dieser Variablen).

Im Forschungsgebiet der ortsbezogenen Informationssysteme (engl. *Location-Awareness*), z.B. im Nexus-Projekt der Universität Stuttgart, in dessen Rahmen diese Arbeit entstanden ist, (näheres ist in [CON99] nachlesbar), spielt die Kenntnis der Position eine tragende Rolle.
Im weiterführenden Forschungsgebiet des Kontext-Bezugs (engl. *Context-Awareness*) werden die Informationen verschiedenster Sensorsysteme miteinbezogen, um den aktuellen Umgebungskontext, der sich einem Benutzer stellt, zu erfassen. Dieser Kontext kann dabei durch unterschiedlichste Informationen, beispielsweise über die Lufttemperatur, die Geschwindigkeit, die Position und viele weitere Klassen von Informationen gebildet werden. Abhängig vom bestimmten Kontext kann ein Computer dann jeweils passende Aktionen durchführen. (Beispiel: Bei Überschreiten einer gewissen Geschwindigkeit wird bei manchen Sportwagen der Firma Porsche automatisch ein Spoiler ausgefahren).
Unter anderem wird auf diesem Gebiet von der „*Future Computing Environments (FCE) Group*" am Georgia Tech-College geforscht, in diesem Zusammenhang wurde hier das sogenannte „*Context Toolkit"* entwickelt (siehe dazu in [16]).
Als mögliche zukünftige Entwicklung kann man sich nun eine Menge von *Interfaces* vorstellen, die über jeweils charakteristische Abfragemethoden Informationen bestimmter Klassen zur Verfügung stellen. Die Funktionalität dieser *Interfaces* wird von den sie implementierenden Klassen bereitgestellt.
Abschließend soll ein beispielhafter Entwurf einer diesbezüglichen Architektur skizziert werden und eine Einordnung der im Rahmen dieser Studienarbeit erstellten Kommunikationsbibliothek, die folgend als „GPSSensor" bezeichnet wird, in diese Architektur vorgenommen werden. Hierbei soll unter dem Begriff „Sensor" eine Einheit verstanden werden, die in der Lage ist, Daten einer bestimmten Menge von Informationsklassen zu messen. So existiert in unten dargestellter Architektur z.B. ein Sensor vom Typ „TimeSensor", der für die Zeitmessung dient.

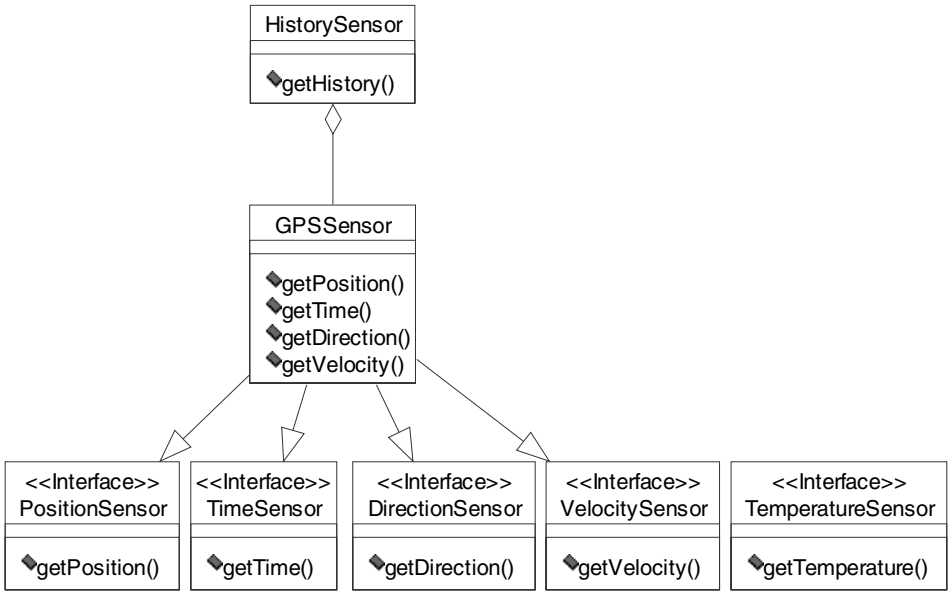


Abbildung 25: UML-Diagramm einer möglichen Sensorarchitektur

Da der Sensor „GPSSensor“ sowohl Position, Zeit, Bewegungsrichtung und Geschwindigkeit bestimmen kann, implementiert dieser dementsprechend die *Interfaces* für Position, Zeit, Richtung und Geschwindigkeit.
Weiterhin soll es möglich sein, Sensoren zu schachteln. So soll z.B. ein „HistorySensor“, der die ermittelten Informationen speichern kann, auf beliebige andere Sensoren aufsetzbar sein. Er stellt damit für all diese Sensoren die Zusatzfunktionalität „Erzeugen einer Historie“ zur Verfügung. In Abbildung 25 wird die beschriebene Sensorarchitektur nochmals dargestellt.

A. Anhang

A.1 Das „GPS“-Package

A.1.1 Die Klasse „NMEASensor“

```
java.lang.Object
   |
+--Sensor.GPS.NMEASensor
```

public abstract class **NMEASensor**
extends java.lang.Object

This abstract class provides methods to handle all general NMEA-Sentences. It will be used by its subclasses, who pass general sentences to their superclass for processing. It is declarated as abstract cause this class will never be instantiated directly but only its subclasses.

Variables:

INITIALIZED
public static final int INITIALIZED

sensor has been successfully initialized

COMPUTING_VALID_FIX
public static final int COMPUTING_VALID_FIX

data collected by the sensor isnt valid yet

RUNNING
public static final int RUNNING

sensor is active and collecting information

STOPPED
public static final int STOPPED

sensor has been stopped

ERROR
public static final int ERROR

there was an error at the initializing, the sensor isnt ready for use

UNDEFINED
public static final int UNDEFINED

default value of the status slot

FULL_DATA
public static final int FULL_DATA

use full set of sentences

SIMPLE_DATA
public static final int SIMPLE_DATA

use a reduced set of sentences

sentencesEnabled
protected int sentencesEnabled

defines which classes of sentences should be used

basicNMEA
protected boolean basicNMEA

Does this sensor use only the basic NMEA data or additional information from proprietary sentences?

newGPSData
protected Sensor.GPSData newGPSData

GPSData object to collect data of the current cycle

currentStatus
protected int currentStatus

Holds the current status of a NMEA-Sensor. Possible values are:
"COMPUTING_VALID_POSITION_FIX", "INITIALIZED", "RUNNING" and some more values.

currentBaudrate
protected int currentBaudrate

baudrate currently used for data transfer

currentGeodeticDatum
protected int currentGeodeticDatum

currently used geodetic datum

defaultBaudrate
protected int defaultBaudrate

default setting for the data transfer rate

defaultGeodeticDatum
protected int defaultGeodeticDatum

default setting for the used geodetic datum

Methods:

getCurrentGPSData
public Sensor.GPSData getCurrentGPSData()
throwsSensorException,NoValidValueException

Returns the currently valid GPSData-object

getCoordinates
public GeographicCoordinate getCoordinates()
throws Sensor.exception.NoValidValueException

Returns the actual position (Latitude, Longitude, height, used geodetic datum

getAccuracy
public double[] getAccuracy()
throws Sensor.exception.NoValidValueException

Returns the actual accuracy (PDOP, HDOP ,VDOP)

getFixQuality
public Java.lang.String[] getFixQuality()
throws Sensor.exception.NoValidValueException

Returns the actual quality of the position fix (Type, quality)

getSentences
public Vector getSentences()

Returns the received sentences of the last cycle and resets the vector there after

setPort
public boolean setPort(Java.lang.String port)
throws Sensor.exception.SensorException

Sets Portname to value of port. Returns false, if port isnt a valid port on this system.

parseSentence
protected Java.lang.String[] parseSentence(Java.lang.String sentence)
throws Sensor.exception.NoValidSentenceException

Is called by the subclasses to parse an incoming sentence. Should only by indirectly called by calling method "processSentence". It saves the transmitted checksum of the sentence in the last field of result[].

processSentence
protected boolean processSentence(Java.lang.String[] data)

Is called to process an incoming NMEA sentence.

update
protected void update()

When called, this method updates the current GPSData and makes it useable by other applications

setUpdated
public void setUpdated(boolean b)

Sets the updated-field to the new value b

getUpdated
public boolean getUpdated()

Returns the value of the updated field

sendData
protected void sendData(Java.lang.String s)
throws Sensor.exception.SensorException

Is used to send data to a connected sensor

init
protected abstract void init()
throws Sensor.exception.SensorException

Is used to initialize a connection with a sensor.

openConnection
protected abstract void openConnection()
throws Sensor.exception.SensorException

Tries to create a connection to the serial port and input/output streams for receiving/sending data to/from the sensor.

startCommunication
public abstract void startCommunication()
throws Sensor.exception.SensorException,
Sensor.exception.NoSuchSensorException

Starts the communication with the sensor.

stopCommunication
public abstract void stopCommunication()

Stops the communication with the sensor

getStatus
public int getStatus()

Returns slot currentStatus

getBasicNMEA
public boolean getBasicNMEA()

Returns slot basicNMEA

checkBaudrate
protected boolean checkBaudrate(int b)

Checks whether a given Baudrate is supported by the sensor board Should be overridden in subclasses of NMEASensor

isRunning
public boolean isRunning()

Returns whether runningThread is currently active or not

getSensorType
public abstract Java.lang.String getSensorType()

Returns the name of the sensor as a String

A.1.2 Die Klasse „Garmin25LPSensor“

```
java.lang.Object
  |
  +--Sensor.GPS.NMEASensor
        |
        +--Sensor.GPS.Garmin25LPSen
```

public class **Garmin25LPSensor**
extends NMEASensor
implements java.lang.Runnable

This class realizes the ability of the interface to "speak" with a Garmin 25LP Sensor Board. This board uses some sentences designed by Garmin Corp. Methods to handle these
sentences are implemented in this class. The 25LP also uses general NMEA-Sentences. If a general sentence is received, the handle method of the superclass is called.

Variables:

currentBaudrate
protected int currentBaudrate

baudrate currently used to transfer data over a serial port

currentGeodeticDatum
protected int currentGeodeticDatum

geodetic datum currently used for position information

Constructors:

Garmin25LPSensor
public Garmin25LPSensor()
throws Sensor.exception.SensorException

This constructor assumes the sensor is connected to the first serial port with 4800 baud. The position sightings use the geodetic datum WGS-84.

Garmin25LPSensor

public Garmin25LPSensor(java.lang.String port)
throws Sensor.exception.SensorException

This constructor assumes the sensor is connected to the serial port specified by the attribute with 4800 baud. The position sightings use the geodetic datum WGS-84.

Garmin25LPSensor
public Garmin25LPSensor(int baud)
throws Sensor.exception.SensorException

This constructor assumes the sensor is connected to the first serial port with the given baud rate. The position sightings use the geodetic datum WGS-84.

Garmin25LPSensor
public Garmin25LPSensor(java.io.File input)
throws java.lang.NullPointerException,
java.io.FileNotFoundException

This constructor assumes that there isnt a real sensor connected and the input source of the sentences will be a file specified by param 'input'. The position sightings use the geodetic
datum WGS-84.

Garmin25LPSensor
public Garmin25LPSensor(java.lang.String port, int baud)
This constructor assumes the sensor is connected to the serial port specified by the attribute with the also specified baudrate. The position sightings use the geodetic datum
WGS-84.

Garmin25LPSensor
public Garmin25LPSensor(int baud,
int geodeticDatum)
throws Sensor.exception.SensorException

This constructor assumes the sensor is connected to the first serial port with the baudrate specified by the attribute. The position sightings use the also specified geodetic datum.

Garmin25LPSensor
public Garmin25LPSensor(java.lang.String port,
int baud,
int geodeticDatum)
throws Sensor.exception.SensorException

This constructor allows to specify the serial port, the baudrate and the geodetic datum used for position sightings by the attributes.

Methods:

setBaudrate
public void setBaudrate(int b)

throws Sensor.exception.SensorException

Sets a new baudrate to the sensor.

setGeodeticDatum
public void setGeodeticDatum(int d)
throws Sensor.exception.SensorException

Sets a new geodetic datum to the sensor.
- is thrown if the given geodetic datum isn't supported by the sensor

checkBaudrate
protected boolean checkBaudrate(int b)

Checks whether a given Baudrate is supported by the sensor board

update
protected void update()

Performs an update, that means data that has been accumulated in super.newGPSData is written to super.currentGPSData. It overrides method update() of class NMEASensor,
but uses this method in its body. Other classes implementing sensors should override super.update() too if they provide some extra data.

processSentence
protected boolean processSentence(java.lang.String[] data)

Overrides method processSentence of class NMEASensor.

init
protected void init()
throws Sensor.exception.SensorException

This method is used to initialize a connection with a sensor. It implements method init() of its abstract superclass NMEASensor

openConnection
protected void openConnection()
throws Sensor.exception.SensorException

Tries to create a connection to the serial port and input/output streams for receiving/sending data to/from the sensor. It implements method openConnection() of its abstract
superclass NMEASensor

startCommunication
public void startCommunication()
throws Sensor.exception.SensorException,
Sensor.exception.NoSuchSensorException

Starts the communication with the sensor. It implements method startCommunication() of class NMEASensor.

stopCommunication
public void stopCommunication()

Stops the communication with the sensor and sets the sensor back to the default values (of the constructor). It implements method stopCommunication() of class NMEASensor.

setSentencesEnabled
public void setSentencesEnabled(int enable)

Sets the sentencesEnabled slot to value enable if enable == FULL_DATA or enable == SIMPLE_DATA, does nothing otherwise.

getError
public double[] getError()
throws Sensor.exception.NoValidValueException

Returns the actual error of the position fix

getVelocity
public double[] getVelocity()
throws Sensor.exception.NoValidValueException

Returns the actual velocity of the sensor

setRunning

public void setRunning(boolean go)

Sets the running field to value go and starts a new thread if was true. Applications using a Garmin25LPSensor shouldnt call this method directly but indirectly by calling method 'startCommunication'.

run
public void run()

Waits for data from the sensor.

getSensorType
public java.lang.String getSensorType()

Returns the name of the sensor as a String, if the sensor is being simulated from a file, this will be noted as "Sim. from file"

getCurrentBaudrate
public int getCurrentBaudrate()

Returns the currently set baudrate.

getCurrentGeodeticDatum
public int getCurrentGeodeticDatum()

Returns the currently set geodetic datum

A.1.3 Die Klasse „GenericNMEASensor“

```
java.lang.Object
  |
  +--Sensor.GPS.NMEASensor
        |
        +--Sensor.GPS.GenericNMEASensor
```

public class **GenericNMEASensor**
extends NMEASensor
implements java.lang.Runnable

Since all NMEA-Sensors are different this class was created to provide the possibilty to use this interface even if your sensor isnt implemented in detail. Thus, this class provides some
basic functionalities that every NMEA-Sensor should be able to fulfill.

Constructors:

GenericNMEASensor
public GenericNMEASensor()
throws Sensor.exception.SensorException

assumes the sensor is connected to the first serial port with 4800 baud. The position sightings use the geodetic datum WGS-84.

GenericNMEASensor
public GenericNMEASensor(java.lang.String port)
throws Sensor.exception.SensorException

assumes the sensor is connected to the serial port specified by the attribute with 4800 baud. The position sightings use the geodetic datum WGS-84.

Methods:

init
protected void init()
throws Sensor.exception.SensorException

Is used to initialize a connection with a sensor. It implements method init() of its abstract superclass NMEASensor

openConnection
protected void openConnection()
throws Sensor.exception.SensorException

Tries to create a connection to the serial port and input/output streams for receiving/sending data to/from the sensor.

startCommunication
public void startCommunication()
throws Sensor.exception.SensorException,
Sensor.exception.NoSuchSensorException

Starts the communication with the sensor. It implements method startCommunication() of class NMEASensor.

stopCommunication
public void stopCommunication()

Stops the communication with the sensor and sets the sensor back to the default values (of the constructor). It implements method stopCommunication() of class NMEASensor.

processSentence
protected boolean processSentence(java.lang.String[] data)

Processes an incoming sentence. For doing this it calls the processSentence method of its superclass

update
protected void update()

Overrides method update of class NMEASensor, but uses this method in its body

setRunning
public void setRunning(boolean go)

Sets the running field to value go and starts a new thread if it was true. Applications using a GenericNMEASensor shouldnt call this method directly but indirectly by calling method 'startCommunication'.

run
public void run()

Waits for data from the sensor.

getSensorType
public java.lang.String getSensorType()

Returns the name of the sensor as a String

A.1.4 Die Klasse HistoryNMEASensor

```
java.lang.Object
  |
  +--Sensor.GPS.HistoryNMEASensor
```

public class **HistoryNMEASensor**
extends java.lang.Object
implements java.lang.Runnable

This class adds the ability of building a history on collected data to a normal NMEASensor

Constructors:

HistoryNMEASensor
public HistoryNMEASensor(NMEASensor sensor)

Creates a new HistoryNMEASensor with an underlaying NMEASensor sensor and a maximum entry value of 5 (default).

HistoryNMEASensor
public HistoryNMEASensor(NMEASensor sensor, int maxEntries)

Creates a new HistoryNMEASensor with an underlaying NMEASensor sensor and a maximum entry value of maxEntries.

Methods:

startListening
public void startListening()

Starts the underlaying sensor and starts the listening HistoryNMEASensor

stopListening
public void stopListening()

Stops the underlaying sensor and stops the listening HistoryNMEASensor

getHistory
public Sensor.GPSData[] getHistory()

returns the history object, that is a collection of max. maxEntries GPSData objects. For information how many GPSData objects are returned in fact, a call of method 'getActEntries()' is intended.

getActEntries
public int getActEntries()

returns how many GPSData objects are returned by a call of method 'getHistory()'.

getHistoryObject
public Sensor.GPSData getHistoryObject(int n)

returns a GPSData object of the history object with index n if n is a valid index and there is an entry in the history with index n, a GPSData object with "empty" values else.

run
public void run()

Implementation of interface java.lang.Runnable.

A.2 Das „Location“-Package

A.2.1 Das Interface „Dimension“

public interface **Dimension**

This interface provides dimension constants and a dimension method.

Variables:

DIM0
public static final int DIM0

Dimension 0

DIM1
public static final int DIM1

Dimension 1

DIM2
public static final int DIM2

Dimension 2

DIM3
public static final int DIM3

Dimension 3

Methods:

getDimension
public int getDimension()

returns the dimension of the object implementing this interface

A.2.2 Die Klasse „Coordinate“

```
java.lang.Object
  |
  +--Sensor.Location.Coordinate
```

public abstract class **Coordinate**
extends java.lang.Object
implements java.lang.Cloneable, Dimension

This class is abstract and provides the basics for coordinates

Variables:

NORTH
public static final double NORTH

the value of the diretion north

NORTHEAST
public static final double NORTHEAST

the value of the direction north-east

EAST
public static final double EAST

the value of the direction east

SOUTHEAST
public static final double SOUTHEAST

the value of the direction south-east

SOUTH
public static final double SOUTH

the value of the direction south

SOUTHWEST
public static final double SOUTHWEST

the value of the direction south-west

WEST

public static final double WEST

the value of the direction west

NORTHWEST
public static final double NORTHWEST

the value of the direction north-west

coordSystem
public CoordSystem coordSystem

underlying coordinate system for coordinate

Constructors:

Coordinate
public Coordinate()

Methods:

equals
public abstract boolean equals(java.lang.Object o)

Compares two Objects for equality

toString
public abstract java.lang.String toString()

returns a string representation of this object
Overrides:

distanceTo
public abstract double distanceTo(Coordinate c)
throws Sensor.exception.NotSupportedException

computes the distance from this coordinate to coordinate c

distance2DTo
public abstract double distance2DTo(Coordinate c)
throws Sensor.exception.NotSupportedException,
Sensor.exception.WrongTypeException

computes the distance from this coordinate to coordinate c, but just in two dimensions (approximation)

directionTo
public abstract Direction directionTo(Coordinate c)
throws Sensor.exception.NotSupportedException

computes the direction from this coordinate to coordinate c

direction2DTo
public abstract Direction direction2DTo(Coordinate c)
throws Sensor.exception.NotSupportedException

computes the direction from this coordinate to coordinate c, but just in two dimensions (approximation)

A.2.3 Die Klasse „CoordSystem“

```
java.lang.Object
  |
  +--Sensor.Location.CoordSystem
```

public abstract class **CoordSystem**
extends java.lang.Object
implements Dimension

This class is abstract and provides the basics for coordinate systems

Constructors:

CoordSystem
public CoordSystem()

Methods:

equals
public abstract boolean equals(java.lang.Object o)

Compares two Objects for equality

toString
public abstract java.lang.String toString()

returns a string representation of this object

getName
public abstract java.lang.String getName()

returns the name of the coordinate system

A.2.4 Die Klasse „Direction“

```
java.lang.Object
  |
  +--Sensor.Location.Direction
```

public abstract class **Direction**
extends java.lang.Object
implements Dimension

This class is abstract and provides the basics for directions

Variables:

geoDat
public int geoDat

underlying geodetic Datum for coordinate

Constructors:

Direction
public Direction()

Methods:

setDimension
public abstract void setDimension(int dim)

sets the dimension of the Direction object

equals
public abstract boolean equals(java.lang.Object o)

Compares two Objects for equality

toString
public abstract java.lang.String toString()

returns a string representation of this object

valueOf
public abstract CoordSystem valueOf(java.lang.String s)
throws Sensor.exception.NotSupportedException

creates upon a string value an instance of this class

A.2.5 Die Klasse „Ellipsoid“

```
java.lang.Object
  |
  +--Sensor.Location.Ellipsoid
```

public class **Ellipsoid**
extends java.lang.Object
implements java.io.Serializable

This class provides ellipsoids (with polar and equator radius)

Variables:

Airy1830
public static final Ellipsoid Airy1830

the ellipsoid Airy 1830

Bessel1841
public static final Ellipsoid Bessel1841

the ellipsoid Bessel 1841

Clarke1866
public static final Ellipsoid Clarke1866

the ellipsoid Clarke 1866

Clarke1880
public static final Ellipsoid Clarke1880

the ellipsoid Clarke 1880

Everest1830
public static final Ellipsoid Everest1830

the ellipsoid Everest 1830

Fischer1960
public static final Ellipsoid Fischer1960

the ellipsoid Fischer 1960

Fischer1968
public static final Ellipsoid Fischer1968

the ellipsoid Fischer 1968

GRS1967
public static final Ellipsoid GRS1967

the ellipsoid GRS 1967

GRS1975
public static final Ellipsoid GRS1975

the ellipsoid GRS 1975

GRS1980
public static final Ellipsoid GRS1980

the ellipsoid GRS 1980

Hough1956
public static final Ellipsoid Hough1956

the ellipsoid Hough 1956

International
public static final Ellipsoid International

the ellipsoid International

Krassovsky1940
public static final Ellipsoid Krassovsky1940

the ellipsoid Krassovsky 1040

SouthAmerican1969
public static final Ellipsoid SouthAmerican1969

the ellipsoid South American 1969

WGS60
public static final Ellipsoid WGS60

the ellipsoid World Geodetic System 1960

WGS66
public static final Ellipsoid WGS66

the ellipsoid World Geodetic System 1966

WGS72
public static final Ellipsoid WGS72

the ellipsoid World Geodetic System 1972

WGS84
public static final Ellipsoid WGS84

the ellipsoid World Geodetic System 1984

Constructors:

Ellipsoid
public Ellipsoid(java.lang.String n,double a, double rf)

creates a new ellipsoid with name, semi major axis and inverse flattening

Ellipsoid
public Ellipsoid(double a, double rf)

creates a new ellipsoid with name = UNKNOWN, semi major axis and inverse flattening

Methods:

equals
public boolean equals(Ellipsoid e)

Compares two Objects for equality

toString
public java.lang.String toString()

returns a string representation of this object

valueOf
public CoordSystem valueOf(java.lang.String s)
throws Sensor.exception.NotSupportedException

creates upon a string value an instance of this class

getName
public java.lang.String getName()

returns the name of the ellipsoid

getSemiMajorAxis
public double getSemiMajorAxis()

returns the semi major axis of the ellipsoid

getInverseFlattening

public double getInverseFlattening()

returns the inverse flattening of the ellipsoid

A.2.6 Die Klasse „GeodeticDatum“

```
java.lang.Object
  |
  +--Sensor.Location.GeodeticDatum
```

public class **GeodeticDatum**
extends java.lang.Object
implements java.io.Serializable

This class provides geodetic datums

Variables:

PREDEFINED
public static final int PREDEFINED

Number of predefined geodetic data

WGS_84
public static final int WGS_84

Constant for Geodectic Datum WGS84

ADINDAN
public static final int ADINDAN

Constant for Geodectic Datum Adindan

ARC1950
public static final int ARC1950

Constant for Geodectic Datum Arc1950

ARC1960
public static final int ARC1960

Constant for Geodectic Datum Arc1960

CAMP_AREA_ASTRO
public static final int CAMP_AREA_ASTRO

Constant for Geodectic Datum Camp Area Astro

EUROPEAN1950
public static final int EUROPEAN1950

Constant for Geodectic Datum European1950

EUROPEAN1979
public static final int EUROPEAN1979

Constant for Geodectic Datum European 1979

FINNISH_NAUTICAL_CHART
public static final int FINNISH_NAUTICAL_CHART

Constant for Geodectic Datum Finnish Nautical Chart

GEODETICDATUM1949
public static final int GEODETICDATUM1949

Constant for Geodectic Datum Geodetic Datum 1949

HONGKONG1963
public static final int HONGKONG1963

Constant for Geodectic Datum Hong Kong 1963

HU_TZU_SHAN
public static final int HU_TZU_SHAN

Constant for Geodectic Datum Hu Tzu Shan

INDIAN_BANGLADESH
public static final int INDIAN_BANGLADESH
Constant for Geodectic Datum Indian Bangladesh

NAD27CONUS
public static final int NAD27CONUS

Constant for Geodectic Datum Nad 27 Conus

NAD_83
public static final int NAD_83

Constant for Geodectic Datum Nad 83

OMAN
public static final int OMAN

Constant for Geodectic Datum Oman

ORD_SUR_GB
public static final int ORD_SUR_GB

Constant for Geodectic Datum Ordnance Survey of Great Britain

PROV_SA1956
public static final int PROV_SA1956

Constant for Geodectic Datum Provisional South American 1956

SA1969
public static final int SA1969

Constant for Geodectic Datum South American 1969

TOKYO
public static final int TOKYO

Constant for Geodectic Datum Tokyo

WGS_72
public static final int WGS_72

Constant for Geodectic Datum WGS72

USER_DEFINED
public static final int USER_DEFINED

Constant for a self defined geodetic datum

UNDEFINED
public static final int UNDEFINED

Constant for an undefined geodetic datum. Used for initialization and to signal that an error had occured.

Adindan
public static final GeodeticDatum Adindan

the geodetic datum Adindan

Arc1950

public static final GeodeticDatum Arc1950

the geodetic datum Arc 1950

Arc1960
public static final GeodeticDatum Arc1960

the geodetic datum Arc 1960

Camp_Area_Astro
public static final GeodeticDatum Camp_Area_Astro

the geodetic datum Camp Area Astro

European1950

public static final GeodeticDatum European1950

the geodetic datum European 1950

European1979
public static final GeodeticDatum European1979

the geodetic datum European 1979

Finnish_Nautical_Chart
public static final GeodeticDatum Finnish_Nautical_Chart

the geodetic datum Finnish Nautical Chart

Geodetic_GeodeticDatum1949
public static final GeodeticDatum Geodetic_GeodeticDatum1949

the geodetic datum Geodetic Datum 1949

Hong_Kong1963
public static final GeodeticDatum Hong_Kong1963

the geodetic datum Hong Kong 1963

Hu_Tzu_Shan
public static final GeodeticDatum Hu_Tzu_Shan

the geodetic datum Hu Tzu Shan

Indian_Bangladesh
public static final GeodeticDatum Indian_Bangladesh

the geodetic datum Indian Bangladesh

NAD27_CONUS
public static final GeodeticDatum NAD27_CONUS

the geodetic datum North American 1927 CONUS

NAD83
public static final GeodeticDatum NAD83

the geodetic datum North American 1983

Oman
public static final GeodeticDatum Oman

the geodetic datum Oman

Ord_Srvy_Grt_Britn
public static final GeodeticDatum Ord_Srvy_Grt_Britn

the geodetic datum Ordnance Survey of Great Britain

Pulkovo1942
public static final GeodeticDatum Pulkovo1942

the geodetic datum Pulkovo 1942

Prov_So_Amrican1956
public static final GeodeticDatum Prov_So_Amrican1956

the geodetic datum Provisional South American 1956

South_American1969
public static final GeodeticDatum South_American1969

the geodetic datum South American 1969

Tokyo
public static final GeodeticDatum Tokyo

the geodetic datum Tokyo

WGS72
public static final GeodeticDatum WGS72

the geodetic datum World Geodetic System 1972

WGS84
public static final GeodeticDatum WGS84

the geodetic datum World Geodetic System 1984

user_defined
public static final GeodeticDatum user_defined

This slot is left free, allowing users to define thier own geodetic datum. Cause it isnt used now it has the same values like WGS84

Constructors:

GeodeticDatum
public GeodeticDatum(java.lang.String n,Ellipsoid e,double dX, double dY,double dZ)

creates a new geodetic datum with name, ellipsoid e, axis values dX, dY and dZ

GeodeticDatum
public GeodeticDatum(Ellipsoid e,double dX,double dY,double dZ)

creates a new geodetic datum with name = UNKNOWN, ellipsoid e, axis values dX, dY and dZ

Methods:

equals
public boolean equals(GeodeticDatum d)

Compares two Objects for equality

toString
public java.lang.String toString()

returns a string representation of this object

valueOf
public CoordSystem valueOf(java.lang.String s)
throws Sensor.exception.NotSupportedException

creates upon a string value an instance of this class

getDX
public double getDX()

returns the deviation of the x-axis

getDY
public double getDY()

returns the deviation of the y-axis

getDZ
public double getDZ()

returns the deviation of the z-axis

getName
public java.lang.String getName()

returns the name of the geodetic datum

getEllipsoid
public Ellipsoid getEllipsoid()

returns the ellipsoid associated with the geodetic datum

getGeodeticDatum
public static GeodeticDatum getGeodeticDatum(int name)
throws Sensor.exception.WrongFormatException

returns the geodetic datum specified by name

getIndex
public static int getIndex(java.lang.String name)
throws Sensor.exception.WrongFormatException

Returns the int constant representing the datum specified by param name.

getList
public static java.lang.String[] getList()

returns a string array with all available predefined geodetic datums

A.2.7 Die Klasse „GeodeticDirection“

```
java.lang.Object
  |
  +--Sensor.Location.Direction
        |
        +--Sensor.Location.GeodeticDirection
```

public class **GeodeticDirection**
extends Direction

This class provides a direction object for the geodetic coordinate system

Constructors:

GeodeticDirection
public GeodeticDirection(double gd, int gs)

creates a new GeodeticDirection with direction and geodetic datum; dimension is set to DIM3

GeodeticDirection
public GeodeticDirection(double gd, int gs, int dim)

creates a new GeodeticDirection with direction geodetic coordinate system and dimension

Methods:

getDimension

public int getDimension()

returns the dimension of the geodetic direction

setDimension
public void setDimension(int dim)

sets the dimension of the Direction object

equals
public boolean equals(java.lang.Object o)

Compares two Objects for equality

toString

public java.lang.String toString()

returns a string representation of this object

valueOf
public CoordSystem valueOf(java.lang.String s)
throws Sensor.exception.NotSupportedException

creates upon a string value an instance of this class

getGeoDat
public int getGeoDat()

returns the geodetic coordinate system of this object

getGeodeticDirection
public double getGeodeticDirection()

returns the geodetic direction

A.2.8 Die Klasse „GeographicCoordinate“

```
java.lang.Object
  |
  +--Sensor.Location.Coordinate
        |
        +--Sensor.Location.GeographicCoordinate
```

public class **GeographicCoordinate**
extends Coordinate
implements Dimension, java.io.Serializable

This class provides a coordinate object for the geographic coordinate system

Variables:

NORTHERN
public static final char NORTHERN

the northern hemisphere

SOUTHERN
public static final char SOUTHERN

the southern hemisphere

EASTERN
public static final char EASTERN

the eastern hemisphere

WESTERN
public static final char WESTERN

the western hemispere

WRONG_COORDINATE
public static double WRONG_COORDINATE

constant for a 'wrong' coordinate

FAC_m_nm
public static int FAC_m_nm

one nautic mile (= 1852 metres)

UNDEFINED_VALUE
public static final double UNDEFINED_VALUE

Constructors:

GeographicCoordinate
public GeographicCoordinate(int cs)

creates a new geographic coordinate with coordinate values (0,0,0) and a geodetic datum

GeographicCoordinate
public GeographicCoordinate(double lat,double lon,double hei, int cs)

creates a new geographic coordinate with coordinate values lat, lon and hei and a geodetic datum positive values mean north(lat), east(lon), respectivley negative values mean south(lat), west(lon), respectivley

GeographicCoordinate
public GeographicCoordinate(double lat,char latHem,double lonchar lonHem,double hei,int cs)

Creates a new geographic coordinate with coordinate values lat, lon, hei, the according hemispheres latHem and lonHem and a geodetic datum

GeographicCoordinate
public GeographicCoordinate(GeographicLatitude lat,GeographicLongitude lon, double hei, int cs)

Creates a new geographic coordinate with coordinate values lat, lon, hei, and a geodetic datum

GeographicCoordinate
public GeographicCoordinate(GeographicLatitude lat,GeographicLongitude lon,int cs)

Creates a new geographic coordinate with coordinate values lat, lon, hei, and a geodetic datum

Methods:

equals
public boolean equals(java.lang.Object o)

Compares two Objects for equality

toString
public java.lang.String toString()

returns a string representation of this object the numbers are rounded to the fifth position after the comma

toString
public java.lang.String toString(boolean verbose)

returns a string representation of this object if verbose is true the description is detailed
Parameters:

valueOf
public static GeographicCoordinate valueOf(java.lang.String s)
throws Sensor.exception.NotSupportedException

creates upon a string value an instance of this class the string value must have the following structure:
NameOfGeodeticDatum(see GeodeticDatum.java) Latitude Longitude Height
Example: 'WGS_84: 23.312 453.2334 4.345'
'WGS_84:23.312 453.2334 4.345'
'WGS_84:23.312,453.2334,4.345'
'WGS_84: 23.312, 453.2334, 4.345'

getDimension
public int getDimension()

returns the dimension of the geographic coordinate

distanceTo
public double distanceTo(Coordinate c)
throws Sensor.exception.NotSupportedException

computes the distance from this coordinate to coordinate c

distance2DTo
public double distance2DTo(Coordinate c)
throws Sensor.exception.NotSupportedException

computes the distance from this geographic Coordinate to geographic Coordinate c, but just in two dimensions (approximation)

directionTo
public Direction directionTo(Coordinate c)
throws Sensor.exception.NotSupportedException

computes the direction from this coordinate to coordinate c

direction2DTo
public Direction direction2DTo(Coordinate c)
throws Sensor.exception.NotSupportedException

computes the direction from this coordinate to coordinate c, but just in two dimensions (approximation)

getLatitude
public double getLatitude()

returns the latitude of the coordinate

getLongitude
public double getLongitude()

returns the longitude of the coordinate

getHeight
public double getHeight()

returns the height of the coordinate

getGeoDat
public int getGeoDat()

returns the geodetic datum of the coordinate

A.2.9 Die Klasse „GeographicLatitude“

```
java.lang.Object
   |
   +--Sensor.Location.GeographicLatitude
```

public class **GeographicLatitude**
extends java.lang.Object

This class provides methods to use a geographic latitude with degrees, minutes and seconds

Variables:

degrees
public int degrees

Holds the degree value of the geographic latitude.

minutes
public int minutes

Holds the minute value of the geographic latitude.

seconds
public double seconds

Holds the second value of the geographic latitude.

hemisphere

public char hemisphere

Holds the hemisphere of the geographic latitude.

Constructors:

GeographicLatitude
public GeographicLatitude()

Creates a new geographic latitude

GeographicLatitude
public GeographicLatitude(int deg, int min,double sec)

Creates a new geographic latitude with the parameters degrees = deg; minutes = min; seconds =sec.

GeographicLatitude
public GeographicLatitude(int deg, int min,double sec,char hem)

Creates a new geographic latitude with the parameters degrees = deg; minutes = min; seconds =sec and hemisphere = hem

Methods:

parseLatitude
public static GeographicLatitude parseLatitude(double lat)

parses a geographic latitude from format ddmm.mmmm to format dd:mm:ss.ss

parseLatitude
public static GeographicLatitude parseLatitude(double lat,char latHem)

parses a geographic latitude from format ddmm.mmmm to format dd:mm:ss.ss

getDoubleValue

public double getDoubleValue()

transforms a geographic latitude with format dd:mm:ss.ss to its corresponding double value with format dd:mmmmmm

toString
public java.lang.String toString()

Gives a String representation of a geographic latitude Format is dd:mm:ss.sss

A.2.10 Die Klasse „GeographicLongitude“

```
java.lang.Object
  |
  +--Sensor.Location.GeographicLongitude
```

public class **GeographicLongitude**
extends java.lang.Object

This class provides methods to use a geographic longitude with degrees, minutes and seconds

Variables:

degrees
public int degrees

Holds the degree value of the geographic longitude.

minutes
public int minutes

Holds the minute value of the geographic longitude.

seconds
public double seconds

Holds the second value of the geographic longitude.

hemisphere
public char hemisphere

Holds the hemisphere of the geographic longitude.

Constructors:

GeographicLongitude
public GeographicLongitude()

Creates a new geographic longitude

GeographicLongitude
public GeographicLongitude(int deg,int min,double sec)

Creates a new geographic longitude with the parameters degrees = deg; minutes = min; seconds =sec.

GeographicLongitude

public GeographicLongitude(int deg,int min, double sec,char hem)

Creates a new geographic longitude with the parameters degrees = deg; minutes = min; seconds =sec and hemisphere = hem

Methods:

parseLongitude
public static GeographicLongitude parseLongitude(double lon)

parses a geographic longitude from format ddmm.mmmm to format dd:mm:ss.sss

parseLongitude
public static GeographicLongitude parseLongitude(double lon,
char lonHem)

parses a geographic longitude from format ddmm.mmmm to format dd:mm:ss.sss

getDoubleValue
public double getDoubleValue()

transforms a geographic longitude with format ddd:mm:ss:sss to its corresponding double value with format ddd:mmmmmm

toString
public java.lang.String toString()

Gives a String representation of a geographic latitude Format is dd:mm:ss.sss

A.3 Die Klasse „GPSData“

```
java.lang.Object
  |
  +--Sensor.GPSData
```

public class **GPSData**
extends java.lang.Object
implements java.lang.Cloneable

This class provides a kind of data-container for use with a GPS-Sensor. The methods that are handling the NMEA-sentences sent by the sensor will put received data in the appropriate
slots. Applications who use the interface can retrieve data they want to use from these slots.

Variables:

EMPTY_INT
public static final int EMPTY_INT

'Empty' value for integer variables of a sensor

EMPTY_CHAR
public static final char EMPTY_CHAR

'Empty' value for character variables of a sensor

EMPTY_STRING
public static final java.lang.String EMPTY_STRING

'Empty' value for string variables of a sensor

Constructors:

GPSData
public GPSData()

Methods:

resetAllValues
public void resetAllValues(boolean basic)

Sets all slots to the default, except some special slots used by sentences that deliver information over more than one cycle. At this moment, only the $GPGSV sentence is implemented.

getUTCTime
public java.lang.String getUTCTime()

returns UTC timestamp of position fix

getUTCDate
public java.lang.String getUTCDate()

returns UTC date of psoition fix

getLatitude
public double getLatitude()

returns geographic latitude of position fix

getLongitude
public double getLongitude()

returns geographic longitude of position fix

getLatHemisphere

public char getLatHemisphere()

returns the geographic Hemisphere of the latitude of the position fix

getLongHemisphere
public char getLongHemisphere()

returns the geographic Hemisphere of the longitude of the position fix

getSpeedOverGround
public double getSpeedOverGround()

returns the speed of the sensor at the current position fix

getCourseMadeGood
public double getCourseMadeGood()

returns the true course of the sensor at the current position fix

getMagCourseMadeGood
public double getMagCourseMadeGood()

returns the magnetic course of the sensor at the current position fix

getMagneticVariation
public double getMagneticVariation()

returns the magnetic variation of the position fix

getMagneticVarDirection
public char getmagneticVarDirection()

returns the direction of the magnetic variation of the position fix

getDataStatus
public char getDataStatus()

returns whether the position data received in the last cycle is valid (A) or not (V)

getCrossTrackError
public double getCrossTrackError()

returns the difference between computed course and actual course

getSteerToCorrect
public char getSteerToCorrect()

returns the direction to steer to correct the actual cross track error

getOriginWaypointID
public int getOriginWaypointID()

returns the ID of the waypoint that is the origin of the current part of the route

getDestinationWaypointID
public int getDestinationWaypointID()

returns the ID of the waypoint that is the destination of the current part of the route

getDestinationWaypointLongitude
public double getDestinationWaypointLongitude()

returns the geographic longitude of the waypoint that is the destination of the current part of the route

getDestWaypointLongHem
public char getDestWaypointLongHem()

returns the geographic Hemisphere of the longitude of the waypoint that is the destination of the current part of the route

getDestinationWaypointLatitude
public double getDestinationWaypointLatitude()

returns the geographic latitude of the waypoint that is the destination of the current part of the route

getDestWaypointLatHem
public char getDestWaypointLatHem()

returns the geographic Hemisphere of the latitude of the waypoint that is the destination of the current part of the route

getRangeToDestination
public double getRangeToDestination()

returns the distance from the current point to the wayoint that is destination of the current part of the route in nautical miles

getTrueBearingToDestination
public double getTrueBearingToDestination()

returns the true bearing to the waypoint that is destination of the current part of the route in degrees

getVelocityTowardsDestination
public double getVelocityTowardsDestination()

returns the velocity towards the waypoint that is destination of the current part of the route in knots

getArrivalAlarm
public char getArrivalAlarm()

returns whether the destination is reached or not

getTrueBearingStartToDestination

public double getTrueBearingStartToDestination()

returns the true bearing from waypoint "start" to waypoint "dest". The Ids of these two waypoint can be retrieved from slots originWaypointID and destinationWaypointID

getMagneticBearingStartToDestination
public double getMagneticBearingStartToDestination()

returns the magnetic bearing from waypoint "start" to waypoint "dest". The Ids of these two waypoint can be retrieved from slots originWaypointID and destinationWaypointID

getAltitude
public double getAltitude()

returns the current altitude of the sensor in metres

getGeoidHeight
public double getGeoidHeight()

returns the height of geoid (mean sea level) above WGS84 reference ellipsoid

getLastDGPSupdate
public double getLastDGPSupdate()

returns the time since last DGPS update in seconds

getDGPSstationID
public int getDGPSstationID()

returns the ID number of the DGPS station that committed the last DGPS-update

getSelectedModeOfFix
public char getSelectedModeOfFix()

returns whether the selection of the type of fix is automatically done (A) or manually (M)

getTypeOfFix
public int getTypeOfFix()

returns the type of the position fix 2 = 2D fix 3 = 3D fix

getFixQuality
public int getFixQuality()

returns the quality of the position fix 0 = no fix possible 1 = GPS fix available 2 = DGPS fix available

getSatellitesUsed
public int getSatellitesUsed()

returns the number of used satellites to compute the position fix

getUsedSatPRNnumbers
public int[] getUsedSatPRNnumbers()

returns the ID numbers of the satellites that were used to compute the position fix

getPDOP
public double getPDOP()

returns the position dilution of precision of the current position fix. It is an indication of the overall precision of the fix

getHDOP
public double getHDOP()

returns the horizontal dilution of precision of the current position fix. It is an indication of the horizontal precision of the fix

getVDOP

public double getVDOP()

returns the vertical dilution of precision of the current position fix. It is an indication of the vertical precision of the fix

getSatellitesInView
public int getSatellitesInView()

returns the number of satellites that are "visible" to the sensor at this time.

getSatPRNnumber
public int[] getSatPRNnumber()

returns the ID numbers of all satellites used for position fixing

getElevation
public int[] getElevation()

returns the elevations of all satellites used for position fixing

getAzimuth
public int[] getAzimuth()

returns the azimuths of all satellites used for position fixing

getSignalStrength
public int[] getSignalStrength()

returns the signal strength of the signals of all satellites used for position fixing

getRouteID
public int getRouteID()

returns the ID of the current route //currently not used

getKmhGroundSpeed
public double getKmhGroundSpeed()

returns the speed of the sensor in kmh

getGeneralWarning
public char getGeneralWarning()

returns if there is a warning flag set by the sensor.

getHorizontalError
public double getHorizontalError()

returns the horizontal error of the position fix in metres. Used by a Garmin proprietary sentence

getVerticalError
public double getVerticalError()

returns the vertical error of the position fix in metres. Used by a Garmin proprietary sentence

getPositionError
public double getPositionError()

returns the overall error of the position fix in metres

getGPSweekNumber
public int getGPSweekNumber()

returns the current GPS week-number (a value between 0 and 1023). Used by a Garmin proprietary sentence

getGPSseconds
public int getGPSseconds()

returns the current GPS second count (a value between 0 and 604799). Used by a Garmin proprietary sentence

getGPSleapSeconds
public int getGPSleapSeconds()

returns the value of the GPS leap second counter. Used by a Garmin proprietary sentence

getVersion
public java.lang.String getVersion()

returns the version of the firmware running on this Garmin sensor Used by a Garmin proprietary sentence

getROMchecksumTest
public char getROMchecksumTest()

returns whether the ROM checksum is true (P) or not (F). Used by a Garmin proprietary sentence

getReceiverFailure
public char getReceiverFailure()

returns whether the receiver has a failure (F) or not (P). Used by a Garmin proprietary sentence

getStoredDataLost
public char getStoredDataLost()

returns whether stored data of the sensor was lost (L) or not (R) Used by a Garmin proprietary sentence

getRealTimeClockLost
public char getRealTimeClockLost()

returns whether the clock of the board is accurate (R) or not (L). Used by a Garmin proprietary sentence

getOscillatorDriftWarning
public char getOscillatorDriftWarning()

returns whether there is an excessive oscilltor drift at the board (F) or not (P). Used by a Garmin proprietary sentence

getDataCollected
public char getDataCollected()

returns whether data is collected by the board (C) or not (null). Used by a Garmin proprietary sentence

getBoardTemperature
public int getBoardTemperature()

returns the current temperature of the board in degrees celcius. Used by a Garmin proprietary sentence

getBoardConfigDataLost
public char getBoardConfigDataLost()

returns whether config data stored by the board is lost (L) or not (R). Used by a Garmin proprietary sentence

getTrueEastVelocity
public double getTrueEastVelocity()

returns the true east velocity component in metres/second. Used by a Garmin proprietary sentence

getTrueNorthVelocity
public double getTrueNorthVelocity()

returns the true north velocity component in metres/second. Used by a Garmin proprietary sentence

getTrueUpVelocity
public double getTrueUpVelocity()

returns the true vertical velocity component in metres/second. Used by a Garmin proprietary sentence

getTDOP
public int getTDOP()

returns the time dilution of precision. Used by a Garmin proprietary sentence

setUTCTime
public void setUTCTime(java.lang.String t)

This method sets slot UTCTime
Parameters:
t - a UTC timestamp

setUTCDate
public void setUTCDate(java.lang.String d)

This method sets slot UTCDate
Parameters:
d - a UTC date

setLatitude
public void setLatitude(double l)

This method sets slot Latitude
Parameters:
l - a geographic latitude

setLongitude
public void setLongitude(double l)

This method sets slot Longitude
Parameters:
l - a geographic longitude

setLatHemisphere
public void setLatHemisphere(char l)

This method sets slot LarHemisphere
Parameters:
l - a Hemisphere of latitude

setLongHemisphere
public void setLongHemisphere(char l)

This method sets slot LongHemnissphere
Parameters:
l - a hemnissphere of longitude

setSpeedOverGround

public void setSpeedOverGround(double s)

This method sets slot speedOverGround
Parameters:
s - the new speed

setCourseMadeGood
public void setCourseMadeGood(double c)

This method sets slot courseMadeGood
Parameters:
c - a new true course

setMagCourseMadeGood
public void setMagCourseMadeGood(double m)

This method sets slot magCourseMadeGood
Parameters:
m - a new magnetic course

setMagneticVariation
public void setMagneticVariation(double v)

This method sets slot magneticVariation
Parameters:

v - a new magnetic variation

setMagneticVarDirection
public void setMagneticVarDirection(char d)

This method sets slot magneticVarDirection
Parameters:
d - a new magnetic variation direction

setDataStatus
public void setDataStatus(char s)

This method sets slot DataStatus
Parameters:
s - the new data status

setCrossTrackError
public void setCrossTrackError(double c)

This method sets slot crossTrackError
Parameters:
c - the new cross track error

setSteerToCorrect
public void setSteerToCorrect(char s)

This method sets slot steerToCorrect
Parameters:
s - the new direction to steer to

setOriginWaypointID
public void setOriginWaypointID(int o)

This method sets slot originWaypointID
Parameters:
o - the new origin waypoint ID

setDestinationWaypointID
public void setDestinationWaypointID(int d)

This method sets slot destinationWaypointID
Parameters:
d - the new destination waypoint ID

setDestinationWaypointLongitude
public void setDestinationWaypointLongitude(double d)

This method sets slot destinationWaypointLongitude
Parameters:
d - the new longitude of the destination waypoint

setDestWaypointLongHem
public void setDestWaypointLongHem(char d)

This method sets slot destWaypointLongHem
Parameters:

d - the new hemnissphere of longitude of the dest. waypoint

setDestinationWaypointLatitude
public void setDestinationWaypointLatitude(double d)

This method sets slot destinationWaypointLatitude
Parameters:
d - the new latitude of the destination waypoint

setDestWaypointLatHem
public void setDestWaypointLatHem(char d)

This method sets slot estWaypointLatHem
Parameters:
d - the new hemnissphere of latitude of the dest. waypoint

setRangeToDestination
public void setRangeToDestination(double r)

This method sets slot rangeToDestination
Parameters:
r - the new distance to the dest. waypoint

setTrueBearingToDestination
public void setTrueBearingToDestination(double t)

This method sets slot trueBearingToDestination
Parameters:
t - the new true bearing to the dest. waypoint

setVelocityTowardsDestination
public void setVelocityTowardsDestination(double v)

This method sets slot velocityTowardsDestination
Parameters:
v - the new velocity towards the dest. waypoint
setArrivalAlarm
public void setArrivalAlarm(char a)

This method sets slot arrivalAlarm
Parameters:
a - the new state of the flag arrival alarm

setTrueBearingStartToDestination
public void setTrueBearingStartToDestination(int t)

This method sets slot trueBearingStartToDestination
Parameters:
t - the new true bearing from the origin waypoint to the dest. waypoint

setMagneticBearingStartToDestination
public void setMagneticBearingStartToDestination(double m)

This method sets slot magneticBearingStartToDestination
Parameters:

m - the new magnetic bearing from the origin waypoint to the dest. waypoint

setAltitude
public void setAltitude(double a)

This method sets slot altitude
Parameters:
a - the new altitude of the sensor

setGeoidHeight
public void setGeoidHeight(double g)

This method sets slot geoidHeight
Parameters:
g - the new geoid height above the WGS84 reference ellipsoid

setLastDGPSupdate
public void setLastDGPSupdate(double l)

This method sets slot lastDGPSupdate
Parameters:
l - the time since the last DGPS update

setDGPSstationID
public void setDGPSstationID(int s)

This method sets slot DGPSstationID
Parameters:
s - the ID of the station that did the last correction.

setSelectedModeOfFix
public void setSelectedModeOfFix(char s)

This method sets slot selectedModeOfFix
Parameters:
s - the new selected mode of fix
setTypeOfFix
public void setTypeOfFix(int t)

This method sets slot typeOfFix
Parameters:
t - the new type of fix

setFixQuality
public void setFixQuality(int q)

This method sets slot fixQuality
Parameters:
q - the new fix quality

setSatellitesUsed
public void setSatellitesUsed(int satellites)

This method sets slot satellitesUsed
Parameters:

satellites - the new number of satellites used for position fix

setUsedSatPRNnumbers
public void setUsedSatPRNnumbers(int[] sat)

This method sets slot usedSatPRNnumbers
Parameters:
sat - the ID-numbers of the satellites used for the fix

setPDOP
public void setPDOP(double p)

This method sets slot PDOP
Parameters:
p - the new value of position dilution of precision

setHDOP
public void setHDOP(double h)

This method sets slot HDOP
Parameters:
h - the new value of horizontal dilution of precision

setVDOP
public void setVDOP(double v)

This method sets slot VDOP
Parameters:
v - the new value of vertical dilution of precision

setSatellitesInView
public void setSatellitesInView(int s)

This method sets slot satellitesInVie
Parameters:
s - the new number of satellites in view

setSatPRNnumber
public void setSatPRNnumber(int[] n,
int current)

This method sets slot satPRNnumber
Parameters:
n - the new set of satellite ID numbers of one cycle

setElevation
public void setElevation(int[] e,
int current)

This method sets slot elevation
Parameters:
e - the new set of elevation of the satellites used for position fix.

setAzimuth
public void setAzimuth(int[] a,
int current)

This method sets slot azimuth
Parameters:
a - the new set of azimuth of the satellites used for position fix

setSignalStrength
public void setSignalStrength(int[] s,
int current)

This method sets slot signalStrength
Parameters:
s - the new set of signal strength of the satellites used for position fix

setRouteID
public void setRouteID(int i)

This method sets slot routeID
Parameters:
i - the new routeID

setKmhGroundSpeed
public void setKmhGroundSpeed(double k)

This method sets slot kmhGroundSpeed
Parameters:
k - the new speed of the sensor in kmh

setGeneralWarning
public void setGeneralWarning(char w)

This method sets slot generalWarning
Parameters:
w - the new state of warning of the sensor

setHorizontalError

public void setHorizontalError(double h)

This method sets slot horizontalError
Parameters:
h - the new horizontal error of the position fix

setVerticalError
public void setVerticalError(double v)

This method sets slot verticalError
Parameters:
v - the new vertical error of the position fix

setPositionError
public void setPositionError(double p)

This method sets slot positionError
Parameters:
p - the new position error of the position fix

setGPSweekNumber
public void setGPSweekNumber(int w)

This method sets slot GPSweekNumber
Parameters:
w - the new GPS week number

setGPSseconds
public void setGPSseconds(int s)

This method sets slot GPSseconds
Parameters:
s - the new value of the GPS seconds count

setGPSleapSeconds
public void setGPSleapSeconds(int l)

This method sets slot GPSleapSeconds
Parameters:
l - the new count of the GPS leap seconds

setVersion
public void setVersion(java.lang.String v)

This method sets slot version
Parameters:
v - the new version of the firmware of the sensor

setROMchecksumTest
public void setROMchecksumTest(char c)

This method sets slot ROMchecksumTest
Parameters:
c - the new value of the flag ROMchecksumTest

setReceiverFailure
public void setReceiverFailure(char f)

This method sets slot receiverFailure
Parameters:
f - the new value of the flag receiverFailure

setStoredDataLost
public void setStoredDataLost(char l)

This method sets slot storedDataLost
Parameters:
l - the new value of the flag storedDataLost

setRealTimeClockLost
public void setRealTimeClockLost(char l)

This method sets slot realTimeClockLost
Parameters:

l - the new value of the flag realTimeClockLost

setOscillatorDriftWarning
public void setOscillatorDriftWarning(char w)

This method sets slot oscillatorDriftWarning
Parameters:
w - the new value of the flag oscillatorDriftWarning

setDataCollected

public void setDataCollected(char c)

This method sets slot dataCollected
Parameters:
c - the new value of the flag dataCollected

setBoardTemperature
public void setBoardTemperature(int t)

This method sets slot boardTemperature
Parameters:
t - the new value of the temperature of the sensor board

setBoardConfigDataLost
public void setBoardConfigDataLost(char l)

This method sets slot boardConfigDataLost
Parameters:
l - the new value of the flag boardConfigDataLost

setTrueEastVelocity
public void setTrueEastVelocity(double e)

This method sets slot trueEastVelocity
Parameters:
e - the new east component of the velocity of the sensor

setTrueNorthVelocity
public void setTrueNorthVelocity(double n)

This method sets slot trueNorthVelocity
Parameters:
n - the new north component of the velocity of the sensor

setTrueUpVelocity
public void setTrueUpVelocity(double u)

This method sets slot trueUpVelocity
Parameters:
u - the new vertical component of the velocity of the sensor

setTDOP
public void setTDOP(int t)

This method sets slot TDOP
Parameters:
t - the new time dilution of precision

duplicate
public GPSData duplicate()

This method is used to clone a GPSData-Object

Literaturverzeichnis

[SSP97] SSP Boatsite: SSP Boatsite Navigation Seminar, http://www.sspboatsite.com/anav.htm, SSP Boatsite, 1997

[WIL97] Williams, E.: Aviation Formulary, http://www.best.com/~williams/avform.htm, Ed Williams Aviation Page, 1997

[TRI97] Trimble Navigation Limited: How GPS works, http://www.trimble.com/gps/fsections/aa_f3.htm, Trimble, 1996 –1997

[MIL00] Milbert, D.G.: Removal of Selective Availability (SA), http://www.ngs.noaa.gov, US Department of Commerce, 2000

[DAN98] Dana, P.H.: The Geographer's Craft Project, http://www.Colorado.EDU/geography/gcraft/notes/notes.html; Department of Geography, The University of Texas, 1998

[SNY93] Snyder, J. P.: Flattening the Earth: Two thousand years of Map Projections, University of Chicago, 1993

[DOC96] US - Department of Defense: NAVSTAR GPS User Equipment Introduction, US – Department of Defense, 1996

[NEX99] Hohl, F., Kubach, U., Leonhardi, U., Rothermel, K., Schwehm, M.: Next Century Challenges: Nexus - An Open Global Infrastructure for Spatial-Aware Applications, Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99), Seattle, 1999

[CON99] Salber, D., Dey, A.K., Abowd, G. D.: The Context Toolkit: Aiding the Development of Context-Enabled Applications, GVU Center, College of Computing, Georgia Institute of Technology, 1999