HOME
HOCHSCHULE MERSEBURG FH
University of Applied Sciences

FACHBEREICH INFORMATIK UND KOMMUNIKATIONS-SYSTEME

BELEGARBEIT

im Fach Mobile Computing (Sommersemester 2010)

Geocaching NavTool

für Microsoft Windows Mobile 2003 SE Pocket-PC-Geräte mit TomTom Wireless MkII GPS Receiver

B.Eng. René SCHWARZ
Matrikelnr.: [redacted] (MIKS09)
E-Mail: mail@rene-schwarz.com

B.Eng. Stefan UTING
Matrikelnr.: [redacted] MIKS09)
E-Mail: stefan.uting@stud.hs-merseburg.de

29. September 2010

Inhaltsverzeichnis

Abstract

Als Semesterarbeit soll im Fach „Mobile Computing“ eine Anwendung für einen Pocket PC entwickelt werden. Da das Thema bzw. die Funktionalität der Software frei wählbar war, hat sich die Gruppe dazu entschieden, ein Navigationstool für das GPS-Spiel „Geocaching“ zu entwickeln. Als Hard- und Softwareumgebung standen dazu ein Acer n50 Pocket PC mit Microsoft Windows Mobile 2003 SE Betriebssystem (Hochschule Merseburg (FH)) und ein TomTom Wireless Mk II GPS Receiver mit Bluetooth-Schnittstelle (privat) zur Verfügung.

Das einfache Tool soll die Teilnahme am Geocaching mit der vorstehenden Hardware ermöglichen, ohne dass weitere (kostenpflichtige) Software notwendig ist.

1 Konzept

Unter dem Namen „Geocaching NavTool“ soll eine einfache Navigationsanwendung für das GPS-Spiel Geocaching entwickelt werden. Derartige Software findet man zwar bereits vielfach (eine Übersicht findet sich z.B. in [5]), diese ist aber zumeist kostenpflichtig oder unterstützt den weit verbreiteten TomTom Wireless MkII GPS Receiver nicht. Als Hardware für diese Zielstellung steht folgende Hardware zur Verfügung:

Geocaching ist eine Art Schatzsuche: mit Hilfe von GPS-Koordinaten werden Verstecke (sog. Caches) gesucht.

- Acer n50 Pocket PC mit Microsoft Windows Mobile 2003 SE Betriebssystem
- TomTom Wireless MkII GPS Receiver mit Bluetooth-Schnittstelle

(a) Acer n50 Pocket PC [7] (b) TomTom GPS Receiver [2]

Abbildung 1: Hardware

Das Geocaching NavTool soll dem Bediener die einfache Eingabe von GPS-Koordinaten ermöglichen und den Benutzer anhand dieser Koordinaten unter Kenntnis seines momentanen Standortes an die gewünschte Position – i.d.R. den Geocache – führen. Die Anwendung funktioniert dabei quasi als digitaler Kompass und Entfernungsmesser. Die notwendigen Standort- und Zustandsdaten (Position, Geschwindigkeit, Richtung etc. pp.) erhält die Software von dem GPS-Empfänger via Bluetooth.

GPS, WGS84 und weitere Begrifflichkeiten werden im Abschnitt 3 näher erläutert.

Ziel ist weiterhin die Entwicklung der Software unter Maßgabe der gültigen Daten- und Formatstandards nach dem geodätischen Referenzsystem WGS84 (World Geodetic System 1984). Dieses System definiert u.a. einen Geoid (ein detailliertes Modell der Erdfigur), ein Koordinatensystem und einen Referenzellipsoiden, die zur Entfernungsberechnung entlag der Erdoberfläche benötigt werden, sowie weitere (für diese Arbeit nicht benötigte) Bestandteile, auf die an dieser Stelle nicht weiter eingegangen werden soll. Das WGS84 ist weit verbreitet und wurde 1989 von der ICAO (International Civil Aviation Organization) als Standard für die Luftfahrt übernommen [11]. Sowohl das Global Positioning System (GPS) als auch die offizielle Geocache-Homepage[1] verwenden das WGS84-Referenzsystem.

Zur Berechnung der Entfernung zweier Punkte auf der Erde muss man die **Oberflächenkrümmung** beachten. WGS84 stellt dafür einen Referenzellipsoiden zur Verfügung.

2 Geocaching

Als Geocaching wird eine moderne Variante der Schnitzeljagd bezeichnet. Dabei werden Container verschiedener Größe, die sogenannten *Caches*, an beliebigen Orten versteckt. Die GPS-Koordinaten des Versteckes werden notiert. Auf speziellen Internet-Plattformen[2] werden diese Koordinaten veröffentlicht und andere Personen können versuchen, den Cache aufgrund der GPS-Koordinaten mit einem GPS-Gerät wiederzufinden.

(a) Inhalt eines Geocaches[3] (b) Travel Bug[4]

Abbildung 2: typische Geocache-Gegenstände

Dabei sind in den Caches meist kleinere Gegenstände sowie ein Logbuch enthalten. In das Logbuch und später auch in den Online-Plattformen kann man den Fund dokumentieren. Auch können Gegenstände in die Cache eingefügt und entnommen werden (diese werden später wieder in einem anderen Cache platziert). Somit sind auch sog. *Travel Bugs* möglich – Gegenstände, die um die ganze Welt reisen.

Es existieren zahlreiche Spielweisen: Oft werden die Zielkoordinaten nicht direkt angegeben, sondern müssen durch ein Rätsel oder andere Aktionen nach und

[1] http://www.geocaching.com
[2] z.B. geocaching.com oder opencaching.de
[3] Bild entnommen von http://de.wikipedia.org/wiki/Datei:Cache_near_%C5%A0pilberk_Castle_%282%29.JPG
[4] Bild entnommen von http://de.wikipedia.org/wiki/Datei:Travel_Bug_in_Hong_Kong.jpg

nach ermittelt werden – dies kann entweder als komplettes Rätsel vorab oder während der Cachesuche über Wegpunkte erfolgen (z.B. „Wieviele Berggipfel sehen Sie von den Koordinaten xyz aus? Die Zahl ist die zweite Ziffer des Längengrads des Caches.“). Viele Caches verbinden die Weg- bzw. Koordinatensuche auch mit einer Tour durch die Umgebung, Landschaft, Sehenswürdigkeiten oder Geschichte des Ortes.

Ebenso zahlreich sind die Varianten der Geocache-Verstecke – von „mit dem Auto erreichbar“ bishin zu „erfordert mehrjährige Bergsteigererfahrung“. Mittlerweile werden auf dem größten Geocaching-Portal, geocaching.com, über 1,2 Millionen Geocaches weltweit geführt. Geocaching verbindet damit das klassische Outdoor-Spiel Schnitzeljagd mit dem modernen Element GPS. Mit – Schätzungen zufolge – über 5 Millionen Spielern weltweit [3] erfreut sich das Geocaching einer großen Beliebtheit.

3 Global Positioning System (GPS) und der GPS-Datenstandard

GPS, GALILEO und GLONASS sind **GNSS**.

Mit der Bezeichnung GNSS (engl. *Global Navigation Satellite System*: globales Navigationssatellitensystem) wird ein System zur Positionsbestimmung und Navigation auf der Erde bezeichnet. Das amerikanische GPS (*Global Positioning System*) ist zum gegenwärtigen Zeitpunkt das einzig voll funktionsfähige GNSS. In den nächsten Jahren werden zwei weitere GNSS zur Verfügung stehen: GALILEO als europäisches und GLONASS als russisches Pendant.

Durch das GPS steht ein GNSS zur Positions- und Zeitbestimmung auf der Erde auch für zivile Nutzung zur Verfügung. Das GPS – ursprünglich für die maritime und militärische Nutzung entwickelt – besteht aus einer Vielzahl von Satelliten, die die Erde in bestimmten Konstellationen umkreisen. Zur Positionsbestimmung mit den zivil üblichen GPS-Empfängern ist der Empfang von mindestens vier Satelliten notwendig (Ermittlung von Längen- und Breitengrad, Höhe und der exakten Zeit). Die Konstellationen der Satellitenumlaufbahnen und die Anzahl der aktiven Satelliten sind dabei so gewählt, dass möglichst von jedem Punkt der Erde aus zu jedem Zeitpunkt mindestens vier Satelliten sichtbar sind.

Ein GPS-Empfänger empfängt die von den Satelliten ausgestrahlten Signale und verarbeitet diese weiter. Am Ende der Verarbeitung wird dem Nutzer eine Vielzahl von Informationen zur aktuellen Position, zur aktuellen Zeit, zum Zustand des GPS u.v.a.m. zur Verfügung gestellt. Auf die Details der Arbeitsweise von GPS/GNSS und die Informationsgewinnung soll an dieser Stelle nicht weiter eingangen werden; der geneigte Leser sei auf andere Literatur, z.B. auf das gut verständliche GPS Compendium von Jean-Marie Zogg [12], verwiesen.

Die **National Marine Electronics Association (NMEA)** erarbeitet Standards für die Schifffahrt.

Der GPS-Empfänger stellt dem Benutzer die gewonnenen Informationen in einem standardisiertem Format, dem NMEA 0183-Datenformat zur Verfügung. Auch der TomTom MkII Wireless GPS Receiver sendet die Daten über die Bluetooth-Verbindung in diesem Format. Dabei werden unterschiedliche Arten von sogenannten GPS-Sätzen in reinen ASCII-Zeichen über die Bluetooth-Verbindung gesendet. Diese enthalten unterschiedliche Informationen über Po-

sition und Lage (Längen- und Breitengrad, Höhe über Geoid und Meer, Bewegungsrichtung, Geschwindigkeit, Fehlerabschätzungen, Anzahl und Zustand der empfangenen Satelliten, Atomuhrzeit etc. pp.). Einen Mitschnitt der Kommunikation mit dem GPS-Empfänger ist in Listing 1 zu sehen.

Listing 1: Kommunikationsmitschnitt der Bluetooth-Verbindung mit dem GPS-Empfänger

```
1  $GPGSA,A,3,27,24,21,15,10,05,18,08,,,,,2.2,1.2,1.8*39
2  $GPRMC,164510.000,A,5121.5157,N,01159.5217,E,0.00,43.78,020410,,*37
3  $GPGGA,164511.000,5121.5157,N,01159.5217,E,1,08,1.2,163.4,M,,,,0000*0E
4  $GPGSA,A,3,27,24,21,15,10,05,18,08,,,,,2.2,1.2,1.8*39
5  $GPRMC,164511.000,A,5121.5157,N,01159.5217,E,0.00,43.78,020410,,*36
6  $GPGGA,164512.000,5121.5157,N,01159.5217,E,1,08,1.2,163.4,M,,,,0000*0D
7  $GPGSA,A,3,27,24,21,15,10,05,18,08,,,,,2.2,1.2,1.8*39
8  $GPRMC,164512.000,A,5121.5157,N,01159.5217,E,0.00,43.78,020410,,*35
9  $GPGGA,164513.000,5121.5157,N,01159.5217,E,1,08,1.2,163.4,M,,,,0000*0C
10 $GPGSA,A,3,27,24,21,15,10,05,18,08,,,,,2.2,1.2,1.8*39
11 $GPRMC,164513.000,A,5121.5157,N,01159.5217,E,0.00,43.78,020410,,*34
12 $GPGGA,164514.000,5121.5157,N,01159.5217,E,1,08,1.2,163.4,M,,,,0000*0B
13 $GPGSA,A,3,27,24,21,15,10,05,18,08,,,,,2.2,1.2,1.8*39
14 $GPGSV,3,1,11,28,60,114,,15,50,295,43,08,50,064,30,05,38,207,30*7F
15 $GPGSV,3,2,11,10,21,184,27,27,18,256,40,07,13,070,,24,10,307,42*7C
```

Es gibt unterschiedliche Arten von **GPS-Sätzen**.

Für die Implementierung im Rahmen dieses Projektes reicht die Analyse der GPRMC- und GPGGA-Sätze aus. Diese liefern die benötigten Positions- und Zustandsangaben. Die Abbildungen 3 und 4 zeigen die Bestandteile der GPRMC- und GPGGA-Sätze. Das Geocaching NavTool zerlegt jeden eingehenden Satz dieser beiden Typen in seine Bestandteile und wertet diese aus.

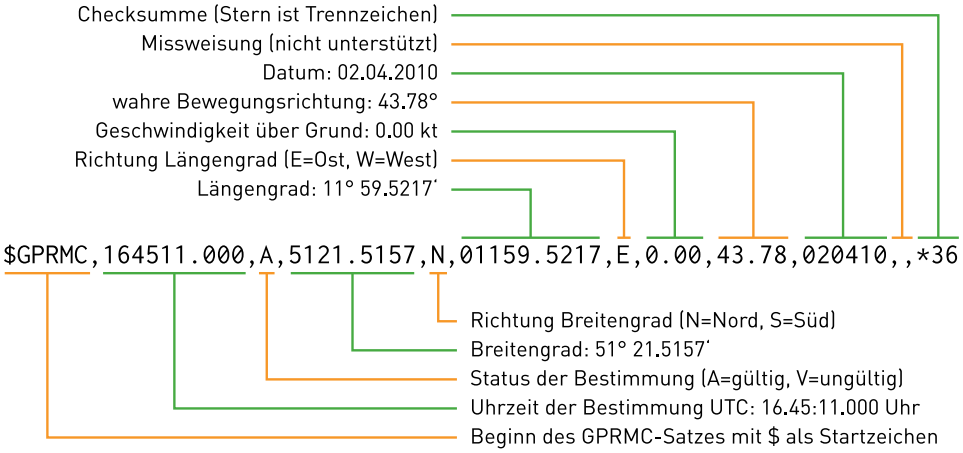

Abbildung 3: Bestandteile eines GPRMC-Satzes (Angaben frei nach [4])

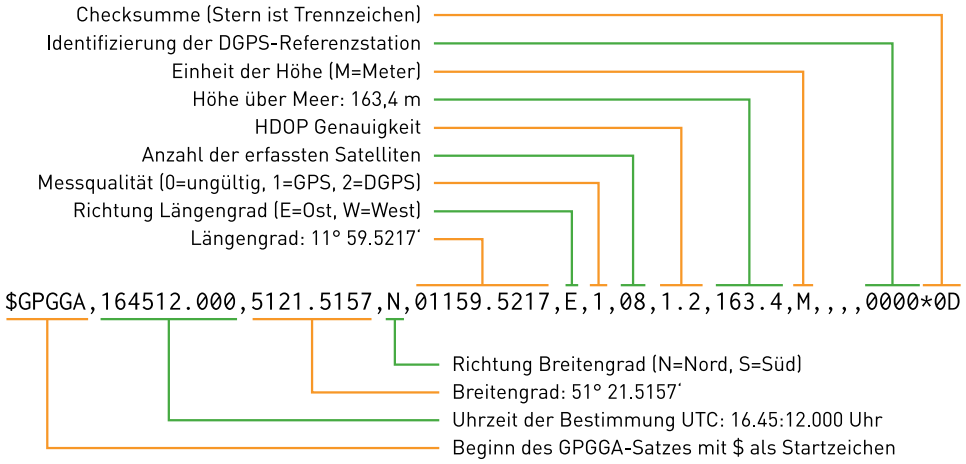

Abbildung 4: Bestandteile eines GPGGA-Satzes (Angaben frei nach [4])

4 Mathematische Grundlagen

Die Software muss verschiedene Berechnungen durchführen, um die GPS-Datensätze auswerten zu können. Dies umfasst die Berechnung der Entfernung zwischen aktueller Position und den Zielkoordinaten sowie der notwendigen Winkeländerung (Laufrichtung) zum Erreichen dieser Koordinaten.

Zudem soll dem Benutzer ein Richtungspfeil dargestellt werden, der die Richtung zum Ziel aufzeigt. Dieser Pfeil wird mit Hilfe eines Polygons mit der C#-Grafik-Funktion `System.Drawing.Graphics.FillPolygon()` gezeichnet; die Koordinaten der Polygoneckpunkte müssen dabei entsprechend transformiert (rotiert) werden.

Vorerst ist die bedeutendste Aufgabe die Erarbeitung einer mathematischen Grundlage für die Entfernungs- und Kursberechnung.

4.1 Sphärische Geometrie

4.1.1 Herleitung einer Kugelparametrisierung

Die Erde wird landläufig als Kugel betrachtet; tatsächlich hat sie aber nicht die Form einer perfekten Kugel, sondern die eines Rotationsellipsoiden. Zunächst beschränken wir uns auf die Kugelform, da diese später recht einfach zu einem Rotationsellipsoiden modifiziert werden kann.

Die **ellipsoide Form der Erde** entsteht durch die Rotation der Erde; dabei wird sie in Normalrichtung ihrer Rotationsachse „auseinandergezogen“.

Gegeben sei eine Kugel mit beliebigen Radius r und dem Mittelpunkt $\vec{O} = (0,0,0)^{\mathrm{T}}$ in einem dreidimensionalen, rechtshändigen, kartesischen Koordinatensystem mit dem Koordinatentripel $\vec{P} = (x,y,z)^{\mathrm{T}}$ für einen beliebigen Punkt $\vec{P}$. Betrachtet man nun zwei voneinander verschiedene Punkte $\vec{P_1}$ und $\vec{P_2}$ auf der Oberfläche der Kugel, so stellt man fest, dass ihre kartesischen Koordinaten ($\vec{P_1} = (x_1,y_1,z_1)^{\mathrm{T}}$ und $\vec{P_2} = (x_2,y_2,z_2)^{\mathrm{T}}$) nur schwer direkt anzugeben sind (vgl. Abb. 5). Auch die Kugel entzieht sich bisher einer Beschreibung in expliziter parametrisierter Form.

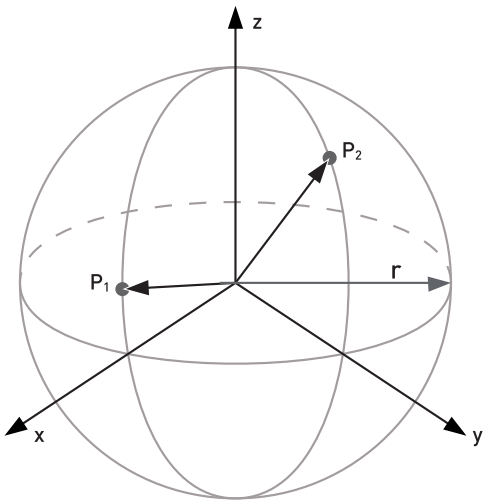

Abbildung 5: Zwei Punkte auf einer Kugeloberfläche mit Mittelpunkt $\vec{O} = (0,0,0)^{\mathrm{T}}$ und Radius r. (Bild: auf Grundlage von [8])

Führt man aber nun einen Winkel $\vartheta \in \mathbb{R}$ zwischen der Applikate[5] und der von Abszisse und Ordinate aufspannten Ebene sowie den Winkel $\varphi \in \mathbb{R}$ zwischen der Abszisse und der zwischen Applikate und Ordinate aufgespannten Ebene ein (vgl. Abb. 6), so kann man die Position eines beliebigen Punktes auf der Kugeloberfläche über die Winkel in Verbindung mit dem Abstand zum Koordinatenursprung $\vec{O}$ angeben. Es wird zusätzlich der Definitionsbereich der beiden Winkel mit $\vartheta \in [0, \pi]$ und $\varphi \in [0, 2\pi[$ vereinbart.

Die Winkel ϑ und φ werden auch als **Polar- und Azimutwinkel** bezeichnet.

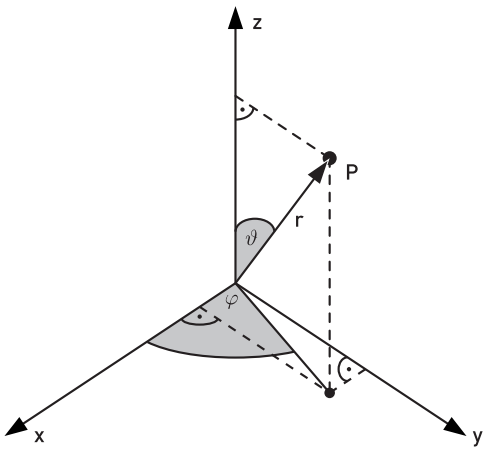

Abbildung 6: Exakte Koordinatenbeschreibung eines Punktes auf der Kugeloberfläche mit Hilfe der Winkel ϑ, φ und dem Radius r.

Es ist nun möglich, einen beliebigen Punkt auf der Oberfläche der Kugel zu beschreiben, ohne die kartesischen Koordinaten zu kennen:

$$\vec{P}_i^{\,\star} = \begin{pmatrix} r \\ \vartheta_i \\ \varphi_i \end{pmatrix} = (r, \vartheta_i, \varphi_i)^{\mathrm{T}} \tag{1}$$

[5] Die „Applikate“ ist der Fachbegriff für die z-Achse eines kartesischen Koordinatensystems (x-Achse: „Abszisse“, y-Achse: Ordinate).

Diese Form der Koordinatenangabe bezeichnet man als räumliche Polarkoordinaten oder Kugelkoordinaten. Der Radius r ist dabei für alle Punkte einer gemeinsamen Kugeloberfläche identisch.

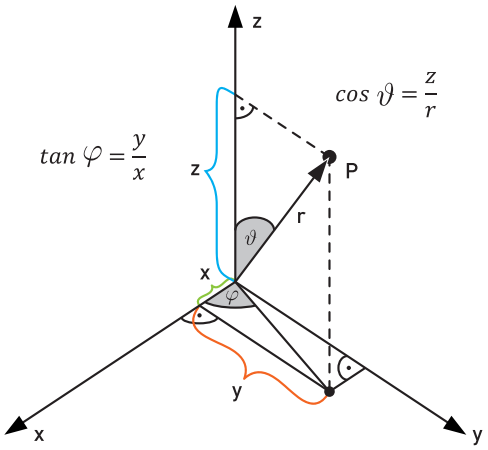

Abbildung 7: Parametrisierung der Kugel.

Um nun die Polarkoordinaten in entsprechende kartesische Koordinaten umrechnen zu können ist eine Parametrisierung der Kugel bezüglich der verwendeten Winkel erforderlich. Dazu betrachten wir Abbildung 7. Der Abstand vom Mittelpunkt (Koordinatenursprung) ist für jeden Punkt $\vec{P}$ gleich – er entspricht dem Radius r des Kreises. Mathematisch formuliert bedeutet dies, dass die euklidische Norm des Vektors $\vec{P}_i$ dem Radius r der Kugel gleicht:

Die **Länge eines Vektors** kann durch seine euklidische Norm bestimmt werden.

$$r = \left\|\vec{P}\right\| = \sqrt{x^2 + y^2 + z^2} \tag{2}$$

Über die ebenen trigonometrischen Funktionen sieht man leicht ein, dass

$$\tan\varphi = \frac{y}{x} \tag{3}$$

und

$$\cos\vartheta = \frac{z}{\sqrt{x^2 + y^2 + z^2}}. \tag{4}$$

Mit Hilfe dieser Vorüberlegungen gelangt man zu der letztlichen Parametrisierung der Kugel: Aus Gleichung 4 folgt

$$\cos\vartheta = \frac{z}{\underbrace{\sqrt{x^2 + y^2 + z^2}}_{=r}}$$

$$\cos\vartheta = \frac{z}{r} \longrightarrow \underline{\underline{z = r\cos\vartheta}}. \tag{5}$$

Mit der Substitution von Gleichung 2 mit der Umformung $x = \frac{y}{\tan\varphi}$ von Gleichung 3 und Gleichung 5 entsteht die Beziehung

Hilfreiche Beziehungen für die Umformungen:

$\tan x = \frac{\sin x}{\cos x}$
$\cos^2 x + \sin^2 x = 1$
$\sqrt{1 - \cos^2 x} = \sin x$

$$r = \sqrt{\frac{y^2}{\tan^2\varphi} + y^2 + r^2\cos^2\vartheta}$$

$$r^2 = y^2\left(\frac{1}{\tan^2\varphi} + 1\right) + r^2\cos^2\vartheta$$

$$
\begin{aligned}
y^2 &= \frac{r^2 - r^2\cos^2\vartheta}{\frac{1}{\tan^2\varphi} + 1} \\
y^2 &= \frac{r^2 - r^2\cos^2\vartheta}{\frac{\cos^2\varphi}{\sin^2\varphi} + 1} \\
y^2 &= \frac{r^2 - r^2\cos^2\vartheta}{\frac{\cos^2\varphi + \sin^2\varphi}{\sin^2\varphi}} \\
y^2 &= (r^2 - r^2\cos^2\vartheta)\sin^2\varphi \\
y^2 &= r^2\sin^2\varphi(1 - \cos^2\vartheta) \\
y &= \sqrt{r^2\sin^2\varphi(1 - \cos^2\vartheta)} \\
y &= r\sin\varphi \cdot \sqrt{(1 - \cos^2\vartheta)} \\
\underline{\underline{y &= r\sin\vartheta\sin\varphi}}. \qquad (6)
\end{aligned}
$$

Über die soeben genutzte lässt sich nun noch die Parametrisierung bezüglich z bestimmen:

$$
\begin{aligned}
x &= \frac{y}{\tan\varphi} \\
&= \frac{r\sin\vartheta\sin\varphi}{\tan\varphi} \\
&= \frac{r\sin\varphi\sin\vartheta\cos\varphi}{\sin\varphi} \\
\underline{\underline{&= r\sin\vartheta\cos\varphi}}. \qquad (7)
\end{aligned}
$$

Mittels den nun vorliegenden Parametrisierungen können sowohl die kartesischen Koordinaten eines in Polarkoordinaten vorliegenden Punktes mit der Funktion

$$
\overrightarrow{P(r,\vartheta,\varphi)} = \begin{pmatrix} x(r,\vartheta,\varphi) \\ y(r,\vartheta,\varphi) \\ z(r,\vartheta) \end{pmatrix} = \begin{pmatrix} r\sin\vartheta\cos\varphi \\ r\sin\vartheta\sin\varphi \\ r\cos\vartheta \end{pmatrix} \qquad (8)
$$

als auch die Parametrisierung einer Kugel $\vec{K}$

$$
\overrightarrow{K(r,\vartheta,\varphi)} = \begin{pmatrix} x(r,\vartheta,\varphi) \\ y(r,\vartheta,\varphi) \\ z(r,\vartheta,\varphi) \end{pmatrix} = \begin{pmatrix} r\sin\vartheta\cos\varphi \\ r\sin\vartheta\sin\varphi \\ r\cos\vartheta \end{pmatrix} \qquad (9)
$$

$$
\text{mit } r = \text{const.}, \vartheta \in [0,\pi], \varphi \in [0, 2\pi[
$$

ausgedrückt werden. Die Winkel ϑ und φ werden als Polar- und Azimutwinkel bezeichnet.

4.1.2 Anwendung auf die Erde

Die hergeleiteten Parametrisierungsgleichungen eignen sich, um die Erde und ihr Gradnetz zu beschreiben. Der Polarwinkel ϑ soll dabei dem Breitengrad, der Azimutwinkel φ dem Längengrad entsprechen. Allerdings werden die Definitionsbereiche der Winkel von unserer Parametrisierung abweichend vereinbart.

Um die in Abschnitt 4.1.1 erarbeitete Kugelparametrisierung auf die Erde an-

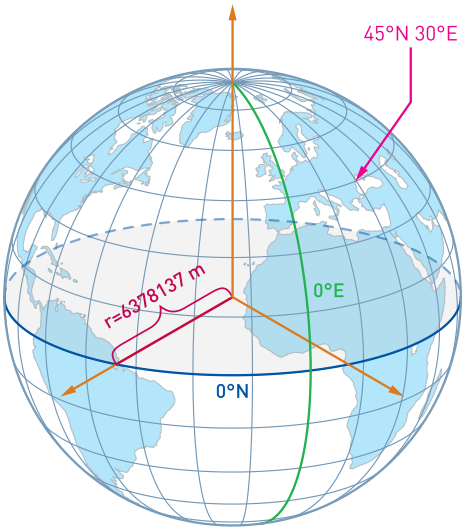

Abbildung 8: Gradnetz der Erde in 15°-Schritten

wenden zu können ist somit eine Anpassung von Gleichung 9 notwendig. Die momentane Parametrisierung geht davon aus, dass ein Vektor mit Winkel ϑ bei einem Wert von $0°$ an der Applikate anliegt; bei der Erde zeigt ein Vektor mit dem Winkel $\vartheta = 0°$ jedoch auf den Äquator (vgl. Abb. 8). Der Winkel muss in der Parametrisierung also modifiziert werden. Zudem laufen die Längengrade der Erde nicht von $0°$ bis $360°$, sondern vom Nullmeridian aus gesehen jeweils von $0°$ bis $180°$ für West- und Ostrichtung. Dies erfordert eine Verschiebung der Winkel in der bisherigen Parametrisierung, so dass das Gradnetz der Erde mit

Die Längengrade der Erde verlaufen von 180° W nach 180° E, die Breitengrade von 90° N bis 90° S. In der gewählten Parametrisierung sind **nördliche/östliche Gradangaben positiv, südliche/westliche negativ**.

$$\overrightarrow{K^{\star}(r_{\oplus}, \vartheta, \varphi)} = \begin{pmatrix} x(r_{\oplus}, \vartheta, \varphi) \\ y(r_{\oplus}, \vartheta, \varphi) \\ z(r_{\oplus}, \vartheta, \varphi) \end{pmatrix} = \begin{pmatrix} r_{\oplus} \sin\left(\frac{\pi}{2} - \vartheta\right) \cos\varphi \\ r_{\oplus} \sin\left(\frac{\pi}{2} - \vartheta\right) \sin\varphi \\ r_{\oplus} \cos\left(\frac{\pi}{2} - \vartheta\right) \end{pmatrix} \tag{10}$$

$$\text{mit } r_{\oplus} = 6\,378\,137\,\text{m}, \vartheta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \varphi \in \,]-\pi, \pi]$$

parametrisiert werden kann (man lege bitte besonderes Augenmerk auf die veränderten Definitionsbereiche).

Der Radius $r_{\oplus}$ bezeichnet dabei den Erdradius der großen Halbachse des Rotationsellipsoiden nach WGS84.

4.2 Entfernungsberechnung

4.2.1 Herleitung einer Basisvariante

Mit der Kugelgleichung (Gleichung 10) kann man den Abstand zwischen zwei Punkten auf der Kugeloberfläche sehr einfach bestimmen. Ein beliebiger Schnitt mit den beiden Punkten $\vec{P_1}$ und $\vec{P_2}$ durch den Mittelpunkt der Kugel ergibt einen Kreis mit Mittelpunkt $(0, 0)^{\mathrm{T}}$ und Radius $r_{\oplus}$. Die kürzeste Entfernung zwischen diesen beiden Punkten ist die Länge des Bogensegments zwischen den beiden Punkten auf diesem Kreis (vgl. Abb. 9).

Das von $\vec{P_1}$ und $\vec{P_2}$ eingeschlossene Kreisbogensegment ist die kürzeste Verbindung der Punkte auf einer Kugeloberfläche und wird als **Orthodrome** bezeichnet.

Die beiden Vektoren der Punkte an der Kugeloberfläche spannen einen gemeinsamen Winkel ε auf, der sich über das Skalarprodukt der beiden Vektoren bestim-

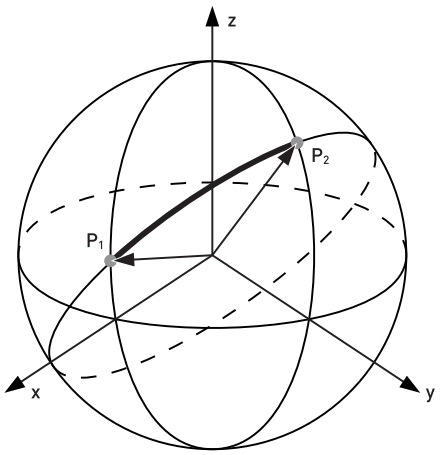

Abbildung 9: Darstellung der Orthodrome für die beiden Punkte $\vec{P}_1$ und $\vec{P}_2$. (Bild: auf Grundlage von [8])

men lässt:

$$\varepsilon = \arccos \frac{\overrightarrow{P_1(r_{♁}, \vartheta_1, \varphi_1)} \cdot \overrightarrow{P_2(r_{♁}, \vartheta_2, \varphi_2)}}{\left\|\overrightarrow{P_1(r_{♁}, \vartheta_1, \varphi_1)}\right\| \cdot \left\|\overrightarrow{P_2(r_{♁}, \vartheta_2, \varphi_2)}\right\|} \tag{11}$$

$$\text{mit } \varepsilon = \sphericalangle\left(\overrightarrow{P_1(r_{♁}, \vartheta_1, \varphi_1)}, \overrightarrow{P_2(r_{♁}, \vartheta_2, \varphi_2)}\right)$$

Das Symbol ♁ ist das **astronomische Symbol für die Erde**.

Es ist bekannt, dass die Länge der Vektoren $\overrightarrow{P_1(r_{♁}, \vartheta_1, \varphi_1)}$ und $\overrightarrow{P_2(r_{♁}, \vartheta_2, \varphi_2)}$ dem Radius $r_{♁}$ entspricht, da die Punkte auf der Kugeloberfläche liegen. Damit gelangt man zu der Gleichung

$$\varepsilon = \arccos \frac{\left\langle \begin{pmatrix} x_1(r_{♁}, \vartheta_1, \varphi_1) \\ y_1(r_{♁}, \vartheta_1, \varphi_1) \\ z_1(r_{♁}, \vartheta_1, \varphi_1) \end{pmatrix}, \begin{pmatrix} x_2(r_{♁}, \vartheta_2, \varphi_2) \\ y_2(r_{♁}, \vartheta_2, \varphi_2) \\ z_2(r_{♁}, \vartheta_2, \varphi_2) \end{pmatrix} \right\rangle}{r_{♁}^2}. \tag{12}$$

Nach dem Ausmultiplizieren und der Vereinfachung der Gleichung entsteht der Ausdruck

$$\begin{aligned} \varepsilon = \arccos \Big[&\sin\left(\frac{\pi}{2} - \vartheta_1\right) \sin\left(\frac{\pi}{2} - \vartheta_2\right) \cos(\varphi_1 - \varphi_2) \\ &+ \cos\left(\frac{\pi}{2} - \vartheta_1\right) \cos\left(\frac{\pi}{2} - \vartheta_2\right) \Big]. \end{aligned} \tag{13}$$

Die Länge des vom Winkel ε eingeschlossenen Kreisbogensegments (vgl. Abb. 10) – und damit die Distanz zwischen den beiden Punkten – lässt sich nun mit

$$\begin{aligned} d(\varepsilon) =& \frac{\pi \cdot r_{♁}}{180°} \cdot \varepsilon \\ =& \frac{\pi \cdot r_{♁}}{180°} \cdot \arccos \Big[\sin\left(\frac{\pi}{2} - \vartheta_1\right) \sin\left(\frac{\pi}{2} - \vartheta_2\right) \cos(\varphi_1 - \varphi_2) \\ &+ \cos\left(\frac{\pi}{2} - \vartheta_1\right) \cos\left(\frac{\pi}{2} - \vartheta_2\right) \Big] \end{aligned} \tag{14}$$

bestimmen.

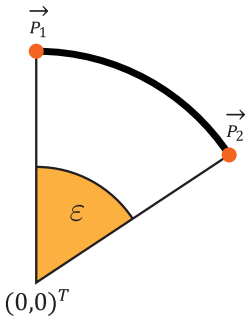

Abbildung 10: Kreisbogensegment zwischen den Punkten $\vec{P_1}$ und $\vec{P_2}$.

4.2.2 Erweiterte Variante

Die hergeleitete Variante verwendet eine Kugel. Mit einer kleinen Modifikation der bereits hergeleiteten Parametrisierung (Gleichung 10) lässt sich auch ein Rotationsellipsoid (und damit die Abplattung der Erde) beschreiben. Die Applikate wird dabei einfach um den Faktor $f_{\oplus}$ gestaucht, so dass sich folgende neue Parametrisierung der Erde ergibt:

$$\overrightarrow{E(r_{\oplus}, f_{\oplus}, \vartheta, \varphi)} = \begin{pmatrix} x(r_{\oplus}, \vartheta, \varphi) \\ y(r_{\oplus}, \vartheta, \varphi) \\ z(r_{\oplus}, f_{\oplus}, \vartheta, \varphi) \end{pmatrix} = \begin{pmatrix} r_{\oplus} \sin\left(\frac{\pi}{2} - \vartheta\right) \cos\varphi \\ r_{\oplus} \sin\left(\frac{\pi}{2} - \vartheta\right) \sin\varphi \\ r_{\oplus} f_{\oplus} \cos\left(\frac{\pi}{2} - \vartheta\right) \end{pmatrix} \tag{15}$$

$$\text{mit } r_{\oplus} = 6\,378\,137\,\text{m}, f_{\oplus} = 1 - \frac{1}{298{,}257\,223\,563}, \vartheta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \varphi \in]-\pi, \pi]$$

$f_{\oplus}$ bezeichnet dabei die Differenz aus 1 und der Abplattung der Erde nach WGS84.

Nun sind die Gleichungen für die resultierende Bogenlänge entlang der Oberfläche zwischen den zwei Punkten nur noch mit sehr viel Aufwand zu bestimmen. Im Jahre 1950 wurde eine Methode veröffentlicht , mit der eine Entfernungsbestimmung auf 50 m genau möglich ist. [1, 6, 10]

Dazu werden zunächst die Hilfsvariablen $F, G, l, S, C, w, D, R, H_1$ und H_2 sowie die (modifizierten) Parameter f und a aus dem WGS84 eingeführt:

$$f \stackrel{!}{=} \frac{1}{298{,}257\,223\,563}, \; a \stackrel{!}{=} \frac{6\,378\,137}{1000} \tag{16}$$

$$F \stackrel{!}{=} \frac{\vartheta_1 + \vartheta_2}{2} \cdot \frac{\pi}{180}, \; G \stackrel{!}{=} \frac{\vartheta_1 - \vartheta_2}{2} \cdot \frac{\pi}{180}, \; l \stackrel{!}{=} \frac{\varphi_1 - \varphi_2}{2} \cdot \frac{\pi}{180} \tag{17}$$

$$S \stackrel{!}{=} (\sin G)^2 \cdot (\cos l)^2 + (\cos F)^2 \cdot (\sin l)^2 \tag{18}$$

$$C \stackrel{!}{=} (\cos G)^2 \cdot (\cos l)^2 + (\sin F)^2 \cdot (\sin l)^2 \tag{19}$$

$$w \stackrel{!}{=} \arctan\sqrt{\frac{S}{C}}, \; D \stackrel{!}{=} 2 \cdot w \cdot a \tag{20}$$

$$R \stackrel{!}{=} \frac{\sqrt{S \cdot C}}{w}, \; H_1 \stackrel{!}{=} \frac{3 \cdot R - 1}{2 \cdot C}, \; H_2 \stackrel{!}{=} \frac{3 \cdot R + 1}{2 \cdot S} \tag{21}$$

Der Abstand d zwischen den beiden Punkten lässt sich dann näherungsweise bestimmen mit

$$d \stackrel{!}{=} D \cdot \left(1 + f \cdot H_1 \cdot (\sin F)^2 \cdot (\cos G)^2 - f \cdot H_2 \cdot (\cos F)^2 \cdot (\sin G)^2\right). \tag{22}$$

Diese Methode ist in das Geocaching NavTool implementiert.

4.3 Kursbestimmung

Das Geocaching NavTool soll dem Benutzer die Richtung des Ziels anzeigen. Dazu ist zunächst die Bestimmung des Kurswinkels zwischen dem derzeitigen Standort und den Zielkoordinaten erforderlich.

Dieser Winkel κ lässt sich über die Beziehung

$$\kappa = \arccos \frac{\cos(\vartheta_A) \cdot \sin(\vartheta_B) - \cos(\varphi_A - \varphi_B) \cdot \cos(\vartheta_B) \cdot \sin(\vartheta_A)}{\sqrt{1 - (\cos(\varphi_A - \varphi_B) \cdot \cos(\vartheta_A) \cdot \cos(\vartheta_B) + \sin(\vartheta_A) \cdot \sin(\vartheta_B))^2}} \tag{23}$$

bestimmen. [10]

Um nun dem Benutzer die Laufrichtung zum Ziel anzuzeigen, muss der Winkel κ von der momentanen Bewegungsrichtung λ, die mit GPS bestimmt wird, abgezogen werden. Somit ergibt sich die notwendige Kursänderung $\kappa^\star$ zu

$$\kappa^\star = \lambda - \arccos \frac{\cos(\vartheta_A) \cdot \sin(\vartheta_B) - \cos(\varphi_A - \varphi_B) \cdot \cos(\vartheta_B) \cdot \sin(\vartheta_A)}{\sqrt{1 - (\cos(\varphi_A - \varphi_B) \cdot \cos(\vartheta_A) \cdot \cos(\vartheta_B) + \sin(\vartheta_A) \cdot \sin(\vartheta_B))^2}}. \tag{24}$$

4.4 Drehungen im zweidimensionalen Raum

Der Pfeil in der Navigationsanzeige des Geocaching NavTools wird über ein Polygon mit drei Eckpunkten definiert. Um den Pfeil zu drehen, ist eine Transformation der Punkte notwendig. Dies geschieht mit Hilfe der allgemein bekannten Rotationsmatrix [9] für jeden der drei Punkte um den Winkel $\kappa^\star$:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos\kappa^\star & -\sin\kappa^\star \\ \sin\kappa^\star & \cos\kappa^\star \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \tag{25}$$

Die Koordinaten $(x_1, y_1)^{\mathrm{T}}$ bezeichnen dabei die ursprünglichen, $(x_2, y_2)^{\mathrm{T}}$ die transformierten (gedrehten) Koordinaten.

5 Beschreibung der Software

5.1 Technische Erläuterung

Das erforderliche Microsoft Compact .NET Framework v3.5 ist dem Softwarepaket beigefügt.

Das Geocaching NavTool wurde in der Programmiersprache *C#* und dem *Microsoft Visual Studio 2009* als Entwicklungsumgebung programmiert. Zur Ausführung ist die vorherige Installation des Microsoft .NET Compact Framework v3.5 auf dem Windows Mobile 2003 SE PDA erforderlich.

Die Anwendung ist als Multithreading-Software konzipiert, um die einzelnen Aufgaben bei gleichzeitigem Erhalt der Benutzerbedienung parallel abarbeiten zu können. Die Anwendung muss im Betrieb folgende Aufgaben übernehmen:

- Darstellung des GUI (engl. „Graphical User Interface“: grafische Benutzeroberfläche) und dessen Interaktionselemente
- Empfangen und Verarbeiten der GPS-Daten aus der Bluetooth-Verbindung mit dem TomTom GPS Receiver

- Visualisieren der dekodierten GPS-Daten für den Benutzer; Zielführung (Richtungs- und Entfernungsanzeige)
- Berechnungen zur Zielführung

Zu diesen Zwecken ist die Anwendung in mehrere Threads gegliedert. Der Haupt-Thread wird beim Starten der Anwendung aufgerufen und erzeugt/erhält die Benutzeroberfläche mit allen Menüs, Buttons, Textboxen etc.

Alsbald der Benutzer die Bluetooth-Verbindung zum GPS-Empfänger initiiert und den GPS-Empfang aktiviert wird der Thread `GetGPSData` gestartet. Dieser Thread liest alle 50 ms den Puffer der Bluetooth-Verbindung aus. Die empfangenen GPS-Sätze werden über die Bildung ihrer Checksummen und den anschließenden Vergleich der übermittelten Checksummen auf ihre Gültigkeit hin überprüft. Alle empfangenen und gültigen GPS-Sätze werden dann je nach Typ entsprechend der in Abschnitt 3 geschilderten Schemata in ihre Bestandteile zerlegt. Die so ermittelten Werte werden über globale Variablen der gesamten Software bekannt gemacht, so dass dieser aller 50 ms aktuelle Daten aus dem GPS-System zur Verfügung stehen.

Gleichzeitig wird der Thread `visualizeGPS` gestartet, der periodisch alle 500 ms die GPS-Anzeigefelder aktualisiert. Der Thread ist dabei nur teilaktiv; die Anzeigeelemente der Zielführung (Richtungs- und Entfernungsanzeige zum Ziel) sind dabei noch inaktiv, denn es liegen noch keine Benutzerangaben über das Ziel vor.

Nachdem der Benutzer gültige Zielkoordinaten eingeben und den Zielführungs-Modus aktiviert hat, wird der Thread `CalculateTarget` gestartet und der Thread `visualizeGPS` in den vollaktiven Zustand versetzt. `CalculateTarget` berechnet nun alle 500 ms die Entfernung und Richtung zu den Zielkoordinaten; `visualizeGPS` stellt nun auch die beiden Informationen in der Benutzeroberfläche dar.

In Abbildung 11 auf Seite 14 ist eine Übersicht der implementierten Threads und die von ihnen aufgerufenen Methoden dargestellt.

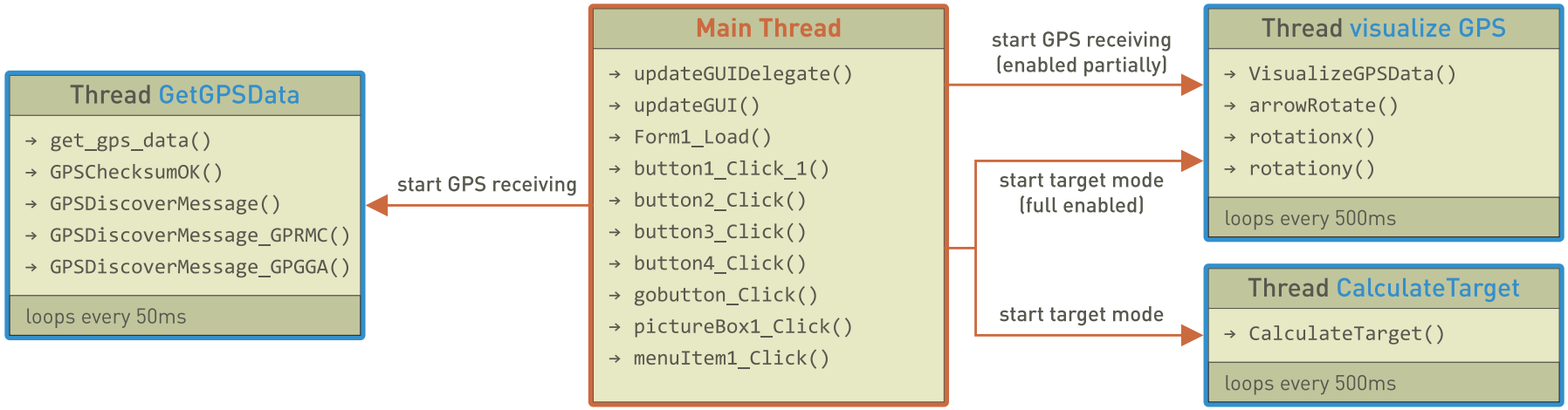

Abbildung 11: Strukturübersicht des Geocaching NavTools.

5.2 Konzept der Benutzerführung

Alsbald der Benutzer die Anwendung startet, erscheint der Willkommensbildschirm (Abbildung 12a). Ein Klick auf das Logo oder die Registerkarte „Setup“ führt den Benutzer in die Einrichtungsoberfläche (Abbildung 12b). Bei dieser hat der Benutzer die Möglichkeit, einen COM-Port als seriellen Kanal für die Bluetooth-Kommunikation zu wählen und zu öffnen; der vom Betriebssystem zugewiesene COM-Port für die Bluetooth-Verbindung kann in den Systemeinstellungen von Windows Mobile ersehen werden.

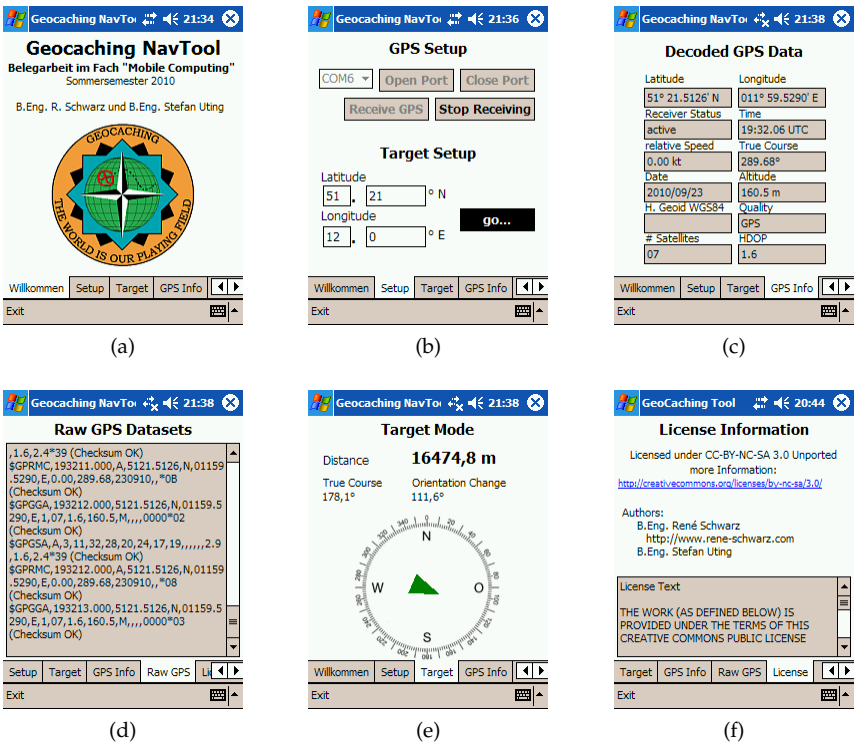

Abbildung 12: GUI des Geocaching NavTools

Nachdem der Port vom Benutzer geöffnet und der TomTom GPS Receiver als Bluetooth-Gerät ausgewählt worden ist, wird der Empfang der GPS-Daten durch den Benutzer aktiviert. Unmittelbar nach der Aktivierung kann der Benutzer alle dekodierten GPS-Informationen in den entsprechenden Menüs ersehen (Abbildungen 12c und 12d).

Direkt unter den Verbindungseinstellungen hat der Benutzer nun die Möglichkeit, die Zielkoordinaten im dezimalen WGS84-Normalformat einzugeben, wie er sie auf der Geocaching-Webseite vorfindet. Ein Klick auf den Go-Button startet dann die Zielführung; der Benutzer wird sofort auf die Navigationsseite geführt, auf der die Entfernung und Richtung zum Ziel angegeben wird (Abbildung 12e).

Der Bediener hat jederzeit die Möglichkeit, die Zielführung zu unterbrechen, z.B. um einen eigenen Geocache zu platzieren und die Koordinaten zu notieren.

Eine detaillierte Bedienungsanleitung ist dieser Ausarbeitung beigefügt.

6 Resultat und Verbesserungsmöglichkeiten

Die Software hat sich im Feldtest als praktikabel und funktionierend erwiesen, wenngleich sie nur die rudimentären Funktionen für das Geocaching (Anzeige des aktuellen Standorts in WGS84-Koordinaten, Eingabe von Zielkoordinaten mit Zielführung) bietet. Die Software birgt großes Erweiterungspotenzial.

Im Moment muss der Benutzer die Zielkoordinaten manuell eingeben; die Geocaching-Webseite bietet auch die Möglichkeit, die Koordinaten sowie eine Beschreibung des Geocaches und weitere damit in Verbindung stehende Daten als XML-Datei herunterzuladen. Mit der Implementierung eines Import-Moduls könnte der Benutzer die heruntergeladene Datei einfach in der Anwendung öffnen und würde sich damit die manuelle Eingabe ersparen. Auch die Darstellung weiterer Daten zum Geocache wäre möglich.

Des Weiteren existieren drei verschiedene Möglichkeiten, GPS-Koordinaten im WGS84-Standardformat anzugeben: Im Dezimalformat (z.B. $51.64213°\,\mathrm{N}\;\;13.1233°\,\mathrm{E}$), Dezimal-Grad-Format (z.B. $51°40.319\,\mathrm{N}\;\;13°64.1315\,\mathrm{E}$) und im Grad-Minuten-Sekunden-Format (z.B. $51°21'43''\,\mathrm{N}\;\;13°25'31''\,\mathrm{E}$). Diese Koordinatenangaben lassen sich recht einfach ineinander umrechnen, so dass ein integriertes Konverter-Tool denkbar wäre.

Abschließend bleibt noch anzumerken, dass die Genauigkeit der vom GPS-Empfänger ermittelten Bewegungsrichtung in hohem Maße von der Bewegungsgeschwindigkeit abhängt. Da GPS selbst bei fixer Position schwankende Angaben zum momentanen Standort liefert, kann die Bewegungsrichtung aufgrund der fälschlich angenommenen Positionsänderung fehlerhaft sein. Bei den im Geocaching üblichen Laufgeschwindigkeiten ergibt sich damit häufig die Problematik, dass die Bewegungsrichtung nur unzureichend genau ermittelt werden kann und damit auch die in der Zielführung angezeigte Richtung zum Ziel fehlerhaft ist. Dies würde sich durch die Verwendung eines GPS-Empfängers mit integriertem digitalen Kompass korrigieren lassen. Ein solcher Empfänger stand der Projektgruppe jedoch zum Zeitpunkt der Projektdurchführung nicht zur Verfügung.

Literatur

[1] BUREAU DES LONGITUDES: *Annuaire du Bureau des Longitudes pour 1950*. Paris, 1950. – p. 145

[2] CHIPSITE: *GPS BLUETooTH - Globalsat BT-338 vs Tomtom GPS receiver mkII*. Internet: http://www.chipsite.pt/forum/viewtopic.php?t=4692, Abruf: 13. September 2010

[3] GROUNDSPEAK, INC.: *Geocaching - The Official Global GPS Cache Hunt Site*. Internet: http://www.geocaching.com/, Abruf: 28. September 2010

[4] KÖHNE, Anja; WÖSSNER, Michael: *Das NMEA-0183 Datenformat*. Internet: http://www.kowoma.de/gps/zusatzerklaerungen/NMEA.htm, Abruf: 28. September 2010

[5] LUHM, Thorsten: *GPS und Geocaching: Software*. Internet: http://www.gps-und-geocaching.de/gps_software.htm, Abruf: 13. September 2010

[6] MEEUS, Jean: *Astronomical Algorithms*. 2nd edition, 2nd printing. Willmann-Bell, Richmond, 2000. – ISBN 0–943396–61–1. – p. 85

[7] TOM'S HARDWARE: *Acer n50 Pocket PC: Introduction*. Internet: http://www.tomshardware.co.uk/acer-n50-pocket-pc-uk,review-1697.html, Abruf: 13. September 2010

[8] WIKIPEDIA: *Datei:Orthodrome_globe.svg — Wikipedia, Die freie Enzyklopädie*. Internet: http://de.wikipedia.org/wiki/Datei:Orthodrome_globe.svg, Abruf: 23. September 2010

[9] WIKIPEDIA: *Drehmatrix — Wikipedia, Die freie Enzyklopädie*. Internet: http://de.wikipedia.org/w/index.php?title=Drehmatrix&oldid=78919066, Abruf: 23. September 2010

[10] WIKIPEDIA: *Orthodrome — Wikipedia, Die freie Enzyklopädie*. Internet: http://de.wikipedia.org/w/index.php?title=Orthodrome&oldid=76345026, Abruf: 23. September 2010

[11] WIKIPEDIA: *World Geodetic System 1984 — Wikipedia, Die freie Enzyklopädie*. Internet: http://de.wikipedia.org/w/index.php?title=World_Geodetic_System_1984&oldid=78583371, Abruf: 13. September 2010

[12] ZOGG, Jean-Marie: *GPS — Essentials of Satellite Navigation*. 1st edition. u-blox AG, 2009. – ISBN 978–3033021396. – also available as free PDF at http://www.u-blox.com/images/downloads/Product_Docs/GPS_Compendium(GPS-X-02007).pdf

A Kommentierter Quellcode

```
1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.ComponentModel;
5  using System.Data;
6  using System.Drawing;
7  using System.Text;
8  using System.Windows.Forms;
9  using System.IO.Ports;
10 using System.Threading;
11
12 namespace GPSPDA2
13 {
14     public partial class Form1 : Form
15     {
16         private SerialPort comport;
17         bool GPSReceivingOnline = false;
18         bool TargetMode = false;
19         Graphics g;
20
21         /* Globale Variablen für die Punkt-zu-Punkt-Berechnung */
22         double b1 = 0;                                  // Breitengrad Punkt 1
23         double l1 = 0;                                  // Längengrad Punkt 1
24         double b2 = 0;                                  // Breitengrad Punkt 2
25         double l2 = 0;                                  // Längengrad Punkt 2
26         double f = 1 / 298.257223563;                   // Abplattung der Erde nach WGS84
27         double a = 6378137 / 1000;                      // (in km umgerechneter) Äquatorradius der Erde
28         double TargetDistance = 0;                      // Initialisierung der Distanz zum Ziel mit 0
29         double TargetCourse = 0;                        // Initialisierung der Richtung zum Ziel mit 0
30
31
32         /* GLOBAL GPS DATASET VARS */
33
34             /* GPGRMC */
35             string DataUhrzeit;                         // Uhrzeit der Bestimmung (UTC)
36             string DataStatus;                          // Status der Bestimmung
37             string DataBreitengrad;                     // Breitengrad mit (Vorzeichen-)Richtung (N=Nord, S=Süd)
38             string DataLaengengrad;                     // Längengrad mit (Vorzeichen-)Richtung
39             string DataGeschwindigkeit;                 // Geschwindigkeit über Grund (Knoten)
40             string DataBewegungsrichtung = "";          // Bewegungsrichtung in Grad (wahr)
41             string DataDatum;                           // Datum
42             string DataMissweisung;                     // Missweisung (mit Richtung)
43
44             /* GPGGA */
45             string DataQualitaetMessung;                // Qualität der Messung
46             string DataAnzahlSatelliten;                // Anzahl der erfassten Satelliten
47             string DataHDOP;                            // HDOP (horizontal dilution of precision) Genauigkeit
48             string DataHoeheUeberMeer;                  // Höhe über Meer (über Geoid) in Metern
49             string DataHoeheGeoidWGS84;                 // Höhe Geoid minus Höhe Ellipsoid (WGS84) in Metern
50
51         /*********************************************************************************************/
52
53
54         delegate void UpdateGUIDelegate(String value);
55
```

```
56          public Form1()
57          {
58              /* Form initalisieren */
59              InitializeComponent();
60          }
61
62          public void UpdateGUI(String value)
63          {
64              /* wird zyklisch ausgeführt, um den Text des GPS-Datenstroms anzuzeigen */
65
66              // textBox1: Anzeige des GPS-Datenstroms (Registerkarte "Raw GPS")
67              // es sollen max. 3000 Zeichen angezeigt werden
68              if (textBox1.Text.Length > 3000)
69              {
70                  textBox1.Text = textBox1.Text.Substring(1000);
71              }
72              textBox1.Text = textBox1.Text + value + "\r\n";
73
74              // Autoscroll der Textbox
75              textBox1.SelectionStart = textBox1.Text.Length;
76              textBox1.ScrollToCaret();
77          }
78
79          private void Form1_Load(object sender, EventArgs e)
80          {
81              /* wird einmalig bei laden des Forms ausgeführt */
82
83              // Auswahlliste der auf dem System verfügbaren COM-Ports erzeugen
84              cmbPort.Items.Clear();
85              foreach (string s in System.IO.Ports.SerialPort.GetPortNames())
86                  cmbPort.Items.Add(s);
87              cmbPort.SelectedIndex = 0;
88
89              // Winkelskala laden (Registerkarte "Target")
90              g = Graphics.FromImage(arrowBox.Image);
91
92              // Hinweisfenster beim Starten der Anwendung anzeigen
93              MessageBox.Show("Diese Anwendung dient als Belegarbeit im Fach \"Mobile Computing\". Bitte "
94              + "lesen Sie die beigefügte Bedienungsanleitung, bevor Sie die Anwendung verwenden.", "Hinweis");
95          }
96
97          private void button1_Click_1(object sender, EventArgs e)
98          {
99              /* Registerkarte "Setup" */
100             /* wird bei Klick auf Button "Open Port" ausgeführt */
101
102             // neuen COM-Port öffnen
103             comport = new SerialPort(this.cmbPort.Text, 2400, Parity.None, 8, StopBits.One);
104             comport.ReadTimeout = 10000;
105             try
106             {
107                 comport.Open();
108             }
109             catch (Exception ex)
110             {
111                 MessageBox.Show("The communication channel to the GPS receiver could not be established. "
112                 + "Probably the GPS receiver is offline or not interconnected via Bluetooth to this system. "
113                 + "\n\nSystem Error: " + ex.Message, "Connection Error");
```

```
114             }
115
116             // Buttons entsprechend aktivieren/deaktivieren, wenn erfolgreich
117             if (comport.IsOpen)
118             {
119                 button1.Enabled = false;
120                 button2.Enabled = true;
121                 button3.Enabled = true;
122                 button4.Enabled = false;
123                 cmbPort.Enabled = false;
124             }
125
126         }
127
128         private void button2_Click(object sender, EventArgs e)
129         {
130             /* Registerkarte "Setup" */
131             /* wird bei Klick auf Button "Close Port" ausgeführt */
132
133             /* COM-Port schließen und Buttons entsprechend aktivieren/deaktivieren */
134             if (comport.IsOpen)
135             {
136                 comport.Close();
137                 button1.Enabled = true;
138                 button2.Enabled = false;
139                 button3.Enabled = false;
140                 button4.Enabled = false;
141                 cmbPort.Enabled = true;
142             }
143         }
144
145         private void button3_Click(object sender, EventArgs e)
146         {
147             /* Registerkarte "Setup" */
148             /* wird bei Klick auf Button "Receive GPS" ausgeführt */
149
150             // global bekanntmachen, dass das GPS receiving online ist
151             GPSReceivingOnline = true;
152             // einen neuen Thread für das GPS receiving deklarieren
153             Thread GetGPSData = new Thread(get_gps_data);
154             // Thread als Background-Thread festlegen
155             GetGPSData.IsBackground = true;
156             // Puffer des COM-Ports leeren
157             comport.DiscardInBuffer();
158             // Thread starten
159             GetGPSData.Start();
160
161             // Buttons entsprechend aktivieren/deaktivieren
162             button2.Enabled = false;
163             button3.Enabled = false;
164             button4.Enabled = true;
165             gobutton.Enabled = true;
166
167             // neuen Thread für die Visualisierung der GPS-Daten deklarieren
168             Thread visualizeGPS = new Thread(VisualizeGPSData);
169             // als Background-Thread festlegen
170             visualizeGPS.IsBackground = true;
171             // und starten
```

```
172             visualizeGPS.Start();
173         }
174 
175         void get_gps_data()
176         {
177             /* Thread GetGPSData */
178             /* Liest alle 50ms den Puffer des COM-Port aus. Veranlasst die Checksummen-
179                Überprüfung der einzelnen GPS-Sätze, das Update der Raw GPS-Textbox und
180                ruft die Funktion GPSDiscoverMessage zur Analyse der GPS-Datensätze auf. */
181 
182             // nur ausführen, wenn das GPS receiving online geschalten ist
183             while (GPSReceivingOnline)
184             {
185                 // COM-Port Puffer auslesen
186                 string comportInput = comport.ReadLine();
187                 // Inhalt in die einzelnen GPS-Sätze auftrennen
188                 comportInput = comportInput.Replace("\r\n", "\n");
189                 string[] gpsSentences = comportInput.Split(new char[] { '\n' });
190 
191                 // Delegate für die Raw GPS-Textbox erzeugen
192                 UpdateGUIDelegate updateGUIDelegate = new UpdateGUIDelegate(UpdateGUI);
193 
194                 // jeden GPS-Satz einzeln behandeln
195                 foreach (string gpsSentence in gpsSentences)
196                 {
197                     // Checksumme des Satzes überprüfen
198                     if (GPSChecksumOK(gpsSentence))
199                     {
200                         // wenn Checksumme ok, dann aktualisiere Textbox im Reiter "Raw GPS"
201                         Invoke(updateGUIDelegate, new String[] { gpsSentence.Replace("\r","")
202                                     + " (Checksum OK)" });
203                         // und analysiere die Inhalte des Satzes
204                         GPSDiscoverMessage(gpsSentence);
205                     }
206                     else
207                     {
208                         // wenn Checksumme nicht in Ordnung, dann aktualisiere Textbox
209                         // im Reiter "Raw GPS" und ignoriere Datensatz
210                         Invoke(updateGUIDelegate, new String[] { "Bad Checksum: " + gpsSentence });
211                     }
212                 }
213 
214                 // 50ms abwarten
215                 Thread.Sleep(50);
216             }
217         }
218 
219         private void button4_Click(object sender, EventArgs e)
220         {
221             /* Registerkarte "Setup" */
222             /* wird bei Klick auf Button "Stop Receiving" ausgeführt */
223 
224             // GPS receiving global ausschalten
225             GPSReceivingOnline = false;
226 
227             // Buttons entsprechend aktivieren/deaktivieren
228             button2.Enabled = true;
229             button3.Enabled = true;
```

```
230             button4.Enabled = false;
231             gobutton.Enabled = false;
232         }
233 
234         private bool GPSChecksumOK(string gpsMessage)
235         {
236             /* Funktion zur Überprüfung der Checksumme eines GPS-Satzes */
237 
238             // erstes Zeichen des Satzes muss ein Dollarzeichen sein
239             if (gpsMessage[0].ToString() == "$")
240             {
241                 // Checksumme mit 0 initialisieren
242                 int checksum = 0;
243                 // Message ist der Teil zwischen Dollarzeichen und Stern-Zeichen
244                 string message = gpsMessage.Substring(1, gpsMessage.IndexOf("*") - 1);
245 
246                 // jeden einzelnen Buchstaben in der Message byteweise XOR'en
247                 foreach (char Character in message)
248                 {
249                     if (checksum == 0)
250                     {
251                         checksum = Convert.ToByte(Character);
252                     }
253                     else
254                     {
255                         checksum = checksum ^ Convert.ToByte(Character);
256                     }
257                 }
258 
259                 // wenn angegebene Checksumme (Zeichen nach dem Stern) und berechnete Checksumme
260                 // übereinstimmen, dann true, andernfalls false
261                 if (checksum.ToString("X2") == gpsMessage.Substring(gpsMessage.IndexOf("*") + 1).Trim())
262                 {
263                     return true;
264                 }
265                 else
266                 {
267                     return false;
268                 }
269             }
270             else
271             {
272                 return false;
273             }
274         }
275 
276         private void GPSDiscoverMessage(string gpsMessage)
277         {
278             /* Diese Funktion veranlasst die Analyse der einzelnen Bestandteile
279                eines GPS-Satzes differenziert je nach seiner Art. */
280 
281             // message type sind die fünf Zeichen nach dem Dollarzeichen
282             string msgType = gpsMessage.Substring(1, 5);
283 
284             // veranlasse die Untersuchung je nach Typ der Message
285             switch (msgType)
286             {
287                 case "GPRMC":
```

```
288                 GPSDiscoverMessage_GPRMC(gpsMessage);
289                 break;
290
291             case "GPGGA":
292                 GPSDiscoverMessage_GPGGA(gpsMessage);
293                 break;
294
295             default:
296                 break;
297         }
298     }
299
300     private void GPSDiscoverMessage_GPRMC(string gpsMessage)
301     {
302         /* Diese Funktion zerlegt eine GPRMC-Nachricht in ihre einzelnen Bestandteile und
303            weist die Werte den vorbereiteten globalen Variablen zu.
304            Weitere Erläuterungen dazu siehe Ausarbeitung. */
305
306         string[] messageParts = gpsMessage.Split(new char[] { ',' });
307
308         if (messageParts[1].Length > 0)
309         {
310             DataUhrzeit = messageParts[1].Substring(0, 2) + ":" + messageParts[1].Substring(2, 2) + "."
311                         + messageParts[1].Substring(4, 2) + " UTC";
312         }
313         if (messageParts[2].Length > 0)
314         {
315             DataStatus = (messageParts[2] == "A") ? "active" : "warning";
316         }
317         if ((messageParts[3].Length > 0) && (messageParts[4].Length > 0))
318         {
319             DataBreitengrad = messageParts[3].Substring(0, 2) + "° " + messageParts[3].Substring(2) + "' "
320                             + messageParts[4];
321             string tmp = messageParts[3].Substring(0, 2) + "."
322                             + Convert.ToString(Convert.ToDouble(messageParts[3].Substring(2)) / 60);
323             if (tmp.IndexOf(",") > 0)
324             {
325                 tmp = tmp.Substring(0, tmp.IndexOf(",") - 1);
326             }
327             b1 = double.Parse(tmp, System.Globalization.CultureInfo.CreateSpecificCulture("en-us"));
328         }
329         if ((messageParts[5].Length > 0) && (messageParts[6].Length > 0))
330         {
331             DataLaengengrad = messageParts[5].Substring(0, 3) + "° " + messageParts[5].Substring(3) + "' "
332                             + messageParts[6];
333             string tmp = messageParts[5].Substring(0, 3) + "."
334                             + Convert.ToString(Convert.ToDouble(messageParts[5].Substring(3)) / 60);
335             if (tmp.IndexOf(",") > 0) { tmp = tmp.Substring(0, tmp.IndexOf(",") - 1); }
336             l1 = double.Parse(tmp, System.Globalization.CultureInfo.CreateSpecificCulture("en-us"));
337         }
338         if (messageParts[7].Length > 0)
339         {
340             DataGeschwindigkeit = messageParts[7] + " kt";
341         }
342         if (messageParts[8].Length > 0)
343         {
344             DataBewegungsrichtung = messageParts[8] + "°";
345         }
```

```
346             if (messageParts[9].Length > 0)
347             {
348                 DataDatum = "20" + messageParts[9].Substring(4, 2) + "/" + messageParts[9].Substring(2, 2) + "/"
349                                  + messageParts[9].Substring(0, 2);
350             }
351             if (messageParts[10].Length > 0)
352             {
353                 DataMissweisung = messageParts[10];
354             }
355         }
356 
357         private void GPSDiscoverMessage_GPGGA(string gpsMessage)
358         {
359             /* Diese Funktion zerlegt eine GPGGA-Nachricht in ihre einzelnen Bestandteile und
360                weist die Werte den vorbereiteten globalen Variablen zu.
361                Weitere Erläuterungen dazu siehe Ausarbeitung. */
362 
363             string[] messageParts = gpsMessage.Split(new char[] { ',' });
364 
365             if (messageParts[1].Length > 0)
366             {
367                 DataUhrzeit = messageParts[1].Substring(0, 2) + ":" + messageParts[1].Substring(2, 2) + "."
368                                  + messageParts[1].Substring(4, 2) + " UTC";
369             }
370             if ((messageParts[2].Length > 0) && (messageParts[3].Length > 0))
371             {
372                 DataBreitengrad = messageParts[2].Substring(0, 2) + "° " + messageParts[2].Substring(2)
373                                  + "' " + messageParts[3];
374             }
375             if ((messageParts[4].Length > 0) && (messageParts[5].Length > 0))
376             {
377                 DataLaengengrad = messageParts[4].Substring(0, 3) + "° " + messageParts[4].Substring(3)
378                                  + "' " + messageParts[5];
379             }
380 
381             if (messageParts[6].Length > 0)
382             {
383                 switch (messageParts[6])
384                 {
385                     case "0":
386                         DataQualitaetMessung = "void";
387                         break;
388 
389                     case "1":
390                         DataQualitaetMessung = "GPS";
391                         break;
392 
393                     case "2":
394                         DataQualitaetMessung = "DGPS";
395                         break;
396 
397                     case "6":
398                         DataQualitaetMessung = "estimated";
399                         break;
400                 }
401             }
402 
403             if (messageParts[7].Length > 0)
```

```
404             {
405                 DataAnzahlSatelliten = messageParts[7];
406             }
407             if (messageParts[8].Length > 0)
408             {
409                 DataHDOP = messageParts[8];
410             }
411             if ((messageParts[9].Length > 0) && (messageParts[10].Length > 0))
412             {
413                 DataHoeheUeberMeer = messageParts[9] + " " + messageParts[10].ToLower();
414             }
415             if ((messageParts[11].Length > 0) && (messageParts[12].Length > 0))
416             {
417                 DataHoeheGeoidWGS84 = messageParts[11] + " " + messageParts[12].ToLower();
418             }
419         }
420 
421         void VisualizeGPSData()
422         {
423             /* Thread visualizeGPS */
424             /* Wird alle 500ms ausgeführt, sofern GPS receiving online ist und aktualisiert
425                alle Anzeigen mit GPS-Daten entsprechend der Werte der globalen Variablen. */
426 
427             // nur ausführen, wenn auch GPS receiving online ist
428             while (GPSReceivingOnline)
429             {
430                 // Änderungen an der GUI
431                 this.BeginInvoke(new Action(() =>
432                 {
433                     // Aktualisierung der einzelnen Textfelder mit den Werten der globalen
434                     // Variablen (glob. Variablen werden vom Thread GetGPSData alle 50ms
435                     // aktualisiert)
436                     this.txtBxBreitengrad.Text = DataBreitengrad;
437                     this.txtBxLaengengrad.Text = DataLaengengrad;
438                     this.txtBxStatus.Text = DataStatus;
439                     this.txtBxUhrzeit.Text = DataUhrzeit;
440                     this.txtBxGeschwindigkeit.Text = DataGeschwindigkeit;
441                     this.txtBxBewegungsrichtung.Text = DataBewegungsrichtung;
442                     this.txtBxDatum.Text = DataDatum;
443 
444                     this.txtBxQualitaetMessung.Text = DataQualitaetMessung;
445                     this.txtBxAnzahlSatelliten.Text = DataAnzahlSatelliten;
446                     this.txtBxHDOP.Text = DataHDOP;
447                     this.txtBxHoeheUeberMeer.Text = DataHoeheUeberMeer;
448                     this.txtBxHoeheGeoidWGS84.Text = DataHoeheGeoidWGS84;
449 
450                     // nur wenn sich der Benutzer in der Zielführung befindet (Zielkoordinaten
451                     // wurden eingegeben und Benutzer hat auf Button "go..." geklickt) wird
452                     // auch die Entfernung und der Kurs zum Ziel aktualisiert
453                     if (TargetMode == true)
454                     {
455                         this.txtTargetDistance.Text = Convert.ToString(TargetDistance) + " m";
456                         this.txtTargetCourse.Text = Convert.ToString(TargetCourse) + "°";
457                     }
458 
459                     // nur wenn GPS-Daten zur Bewegungsrichtung vorliegen
460                     if (DataBewegungsrichtung.Length > 0)
461                     {
```

```
462                     // Winkel aus GPS-Daten auslesen
463                     float angle = float.Parse(DataBewegungsrichtung.Substring(0,
464                                     DataBewegungsrichtung.Length-1),
465                                     System.Globalization.CultureInfo.CreateSpecificCulture("en-us"));
466
467                     // nur wenn Zielführung aktiv
468                     if (TargetMode == true)
469                     {
470                         // Kurs ermitteln (siehe Ausarbeitung)
471                         double CourseChange = Math.Round(Convert.ToDouble(angle) - TargetCourse, 1);
472                         this.txtTargetChange.Text = Convert.ToString(CourseChange) + "°";
473                         angle = (float)CourseChange;
474                     }
475                     // Aktualisierung des Richtungspfeils mit dem soeben errechneten Kurswinkel
476                     arrowRotate(angle);
477                 }
478             }));
479
480             // 500ms abwarten
481             Thread.Sleep(500);
482         }
483     }
484
485     private void gobutton_Click(object sender, EventArgs e)
486     {
487         /* Registerkarte "Setup" */
488         /* wird bei Klick auf Button "go..." bzw. "stop..." ausgeführt */
489
490         // wenn Zielführung inaktiv (bisher noch nicht auf "go..." geklickt)
491         if (TargetMode == false)
492         {
493             this.gobutton.Text = "stop...";     // aus go-Button stop-Button machen :o)
494             int b21 = 0;                        // Zielbreitengrad (Vorkommastellen)
495             int b22 = 0;                        // Zielbreitengrad (Nachkommastellen)
496             int l21 = 0;                        // Ziellängengrad (Vorkommastellen)
497             int l22 = 0;                        // Ziellängengrad (Nachkommastellen)
498
499             // Benutzereingaben konvertieren und den Variablen zuweisen
500             // hier steckt noch ein Fehler: werden in den Nachkommastellen führende Nullen
501             // eingegeben, so werden diese abgeschnitten. Bsp: 12.0452 wird zu 12.452
502             // verhindern kann man das, indem man eine explizite Konvertierung zu einer Float
503             // vornimmt und nicht erst den Schritt über int32 geht...
504             if (txtTargetLat1.Text.Length > 0)  { b21 = Convert.ToInt32(this.txtTargetLat1.Text);  }
505             if (txtTargetLat2.Text.Length > 0)  { b22 = Convert.ToInt32(this.txtTargetLat2.Text);  }
506             if (txtTargetLong1.Text.Length > 0) { l21 = Convert.ToInt32(this.txtTargetLong1.Text); }
507             if (txtTargetLong2.Text.Length > 0) { l22 = Convert.ToInt32(this.txtTargetLong2.Text); }
508
509             // Zielbreiten- und -längengrad zusammensetzen
510             b2 = double.Parse(Convert.ToString(b21) + "." + Convert.ToString(b22),
511                         System.Globalization.CultureInfo.CreateSpecificCulture("en-us"));
512             l2 = double.Parse(Convert.ToString(l21) + "." + Convert.ToString(l22),
513                         System.Globalization.CultureInfo.CreateSpecificCulture("en-us"));
514
515             // Zielführung aktivieren
516             TargetMode = true;
517             // neuen Thread für die Zielführungs-Berechnungen deklarieren
518             Thread CalculateTarget = new Thread(calculate_target);
519             // als Background-Thread festlegen
```

```
520                 CalculateTarget.IsBackground = true;
521                 // und starten
522                 CalculateTarget.Start();
523                 // auf Tab "Setup" wechseln
524                 tabControl1.SelectedIndex = 2;
525             }
526             else
527             {
528                 this.gobutton.Text = "go...";               // Stop-Button in einen Go-Button umwandeln :)
529                 TargetMode = false;                         // Zielführung deaktivieren
530             }
531         }
532 
533         void calculate_target()
534         {
535             /* Thread CalculateTarget */
536             /* Führt alle 500ms die Berechnungen für die Entfernung und dem Kurswinkel zum Ziel aus
537                und weist den entsprechenden globalen Variablen die Werte zu.
538                Weitere Erläuterungen in der Ausarbeitung. */
539 
540             while (TargetMode)
541             {
542                 double F = ((b1 + b2) / 2) * (Math.PI / 180);
543                 double G = ((b1 - b2) / 2) * (Math.PI / 180);
544                 double l = ((l1 - l2) / 2) * (Math.PI / 180);
545                 double S = Math.Pow(Math.Sin(G), 2) * Math.Pow(Math.Cos(l), 2) + Math.Pow(Math.Cos(F), 2)
546                             * Math.Pow(Math.Sin(l), 2);
547                 double Co = Math.Pow(Math.Cos(G), 2) * Math.Pow(Math.Cos(l), 2) + Math.Pow(Math.Sin(F), 2)
548                             * Math.Pow(Math.Sin(l), 2);
549                 double omega = Math.Atan(Math.Sqrt(S / Co));
550                 double Dh = 2 * omega * a;
551                 double R = (Math.Sqrt(S * Co)) / omega;
552                 double H1 = (3 * R - 1) / (2 * Co);
553                 double H2 = (3 * R + 1) / (2 * S);
554                 double s = Dh * (1 + f * H1 * Math.Pow(Math.Sin(F), 2) * Math.Pow(Math.Cos(G), 2) - f * H2
555                             * Math.Pow(Math.Cos(F), 2) * Math.Pow(Math.Sin(G), 2));
556                 double zeta = Math.Acos(Math.Sin(b1 * (Math.PI / 180)) * Math.Sin(b2 * (Math.PI / 180))
557                             + Math.Cos(b1 * (Math.PI / 180)) * Math.Cos(b2 * (Math.PI / 180))
558                             * Math.Cos((l2 - l1) * (Math.PI / 180))) * (180 / Math.PI);
559                 double alpha = Math.Acos((Math.Sin(b2 * (Math.PI / 180)) - Math.Sin(b1 * (Math.PI / 180))
560                             * Math.Cos(zeta * (Math.PI / 180))) / (Math.Cos(b1 * (Math.PI / 180))
561                             * Math.Sin(zeta * (Math.PI / 180)))) * (180 / Math.PI);
562 
563                 TargetDistance = Math.Round(s*1000,1);
564                 TargetCourse = Math.Round(alpha, 1);
565 
566                 Thread.Sleep(500);
567             }
568         }
569 
570         private void pictureBox1_Click(object sender, EventArgs e)
571         {
572             /* Registerkarte "Welcome" */
573             /* wird bei Klick auf das Logo ausgeführt */
574 
575             // auf den nächsten Tab wechseln
576             tabControl1.SelectedIndex = 1;
577         }
```

```
578
579         private void arrowRotate(float angle)
580         {
581             /* Funktion zur Rotation des Pfeils zur Zielführung */
582
583             // PictureBox löschen
584             g.Clear(Color.White);
585
586             // Pinsel erzeugen
587             SolidBrush greenBrush = new SolidBrush(Color.Green);
588
589             // Polygonpunkte definieren
590             // (vorher die Transformationsfunktionen aufrufen)
591             Point[] curvePoints =
592              {
593                  new Point(rotationx(15, 40, angle), rotationy(15, 40, angle)),
594                  new Point(rotationx(25, 10, angle), rotationy(25, 10, angle)),
595                  new Point(rotationx(35, 40, angle), rotationy(35, 40, angle))
596              };
597
598             // gefülltes Polygon zeichnen
599             g.FillPolygon(greenBrush, curvePoints);
600         }
601
602         private int rotationx(int x, int y, float angle)
603         {
604             /* Gibt die um den Winkel angle rotierte x-Koordinate zurück. */
605
606             return Convert.ToInt32(Math.Round((x - 25) * Math.Cos(angle * (Math.PI / 180)) - (y - 25)
607                             * Math.Sin(angle * (Math.PI / 180)), 0)) + 25;
608         }
609
610         private int rotationy(int x, int y, float angle)
611         {
612             /* Gibt die um den Winkel angle rotierte y-Koordinate zurück. */
613
614             return Convert.ToInt32(Math.Round((x - 25) * Math.Sin(angle * (Math.PI / 180)) + (y - 25)
615                             * Math.Cos(angle * (Math.PI / 180)), 0)) + 25;
616         }
617
618         private void menuItem1_Click(object sender, EventArgs e)
619         {
620             /* Exit-Button */
621
622             // wenn GPS receiving noch online ist, Schließen des Programms
623             // verhindern
624             // Grund: Benutzer könnte versehentlich geklickt haben
625             if (GPSReceivingOnline)
626             {
627                 MessageBox.Show("Please stop GPS receiving first.", "Denied");
628                 // auf den ersten Tab wechseln
629                 tabControl1.SelectedIndex = 1;
630             }
631             else
632             {
633                 if (button2.Enabled)
634                 {
635                     // COM-Port schließen, falls noch offen
```

```
636                     // Begründung: Ist der COM-Port noch offen, ist er für andere
637                     // Anwendungen blockiert.
638                     comport.Close();
639                 }
640                 // Anwendung beenden
641                 Application.Exit();
642             }
643         }
644 
645     }
646 }
```