TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Professur Theoretische Informatik und Informationssicherheit

Bachelorarbeit

Implementierung einer webbasierten Anwendung zur Routenplanung und -optimierung anhand einer Fallstudie

vorgelegt von: Martin Rein
geboren am 21. August 1991 in Zschopau

im Studiengang: Bachelor Informatik

Betreuer: Prof. Dr. Hanno Lefmann,
Michael Schräber (M.Sc.)

Tag der Ausgabe: 01.08.2016
Tag der Abgabe: 05.12.2016

Chemnitz, den 05. Dezember 2016

Inhaltsverzeichnis

Abkürzungsverzeichnis

EPSG	European Petroleum Survey Group
KMU	kleine und mittelständische Unternehmen
lat	Latitude (geographische Breite)
lon	Longitude (geographische Länge)
GPS	Global Positioning System
OL	OpenLayers (JavaScript-Bibliothek)
OSM	OpenStreetMap (JavaScript-Bibliothek)
POI	Point (Punkt in einer Karte)
px	Pixel
SRID	Spatial Reference Identifier
StVO	Straßenverkehrs-Ordnung
TSP	Problem Travelling Salesman

Bilderverzeichnis

Tabellenverzeichnis

1. Einleitung

Unternehmen mit Mitarbeitern im Außendienst benötigen heutzutage zur Disposition und Kontrolle ihrer Mitarbeiter eine Softwareunterstützung, um am Markt wettbewerbsfähig zu bleiben. Dazu zählen im KMU-Segment insbesondere Speditionen und Lieferdienste, ambulante Pflegedienste, aber auch Bau- und Handwerksgewerbe. Dabei spielt entweder die kürzeste oder schnellste Strecke eine Rolle. Faktoren wie die Länge der Strecke, die dafür benötigte Fahrzeit und auch zusätzlich die Verkehrsinformationen (Baustellen, Stau, usw.) sind von wichtiger Bedeutung, da diese Informationen für die Berechnung wichtig sind. Ein Blick auf eine Karte reicht meistens nicht aus, um die Länge und Fahrzeit der Strecke zu ermitteln. Mittels heutiger moderner Technik wird die Verbindung zwischen zwei Punkten relativ schnell berechnet und dem Nutzer angezeigt.
Noch vor einigen Jahren musste man auf ein analoges Medium (Atlas, Straßenatlas und Stadtkarten) zurückgreifen. Die Route von Ort A zu Ort B war zu berechnen, aber auch nur mit groben Werten. Wenn zwischen dem Ort A und B sich noch die Zwischenziele C, D, E und F befanden, dann war es sehr schwierig, einen effizienten Weg zu finden und Angaben zur Gesamtlänge der Route und Fahrzeit zu machen. Auch auf weitere nützliche Zusatzinformationen wie Verkehrsinformationen musste man verzichten und sich auf die aktuelle Meldung vom Rundfunk oder Automobilklub verlassen. Fahrer mit CB-Funk im Fahrzeug waren und sind immer noch in der Lage, die Verkehrssituationen auf der Straße mit ihren Berufskollegen auszutauschen.
In vielen Transportfahrzeugen sind mittlerweile GPS-fähige Endgeräte mit Verbindung zum mobilen Telekommunikationsnetz verbaut, so dass zum Beispiel die Geschäftsführung und Disponenten eines Unternehmens die Fahrzeuge orten und in der Lage sind, mit den Fahrzeugführern zu kommunizieren und Anweisungen zu geben. Auch Aufzeichnungen von Routenverläufen haben an Interesse gewonnen und dienen zur Auswertung vom Verlauf der absolvierten Transporte und Überprüfung der Fahrzeugführung.
Ziel dieser Bachelorarbeit ist es, ein Anwendungssystem zu entwickeln, die das Planen und das Verfolgen von Routen ermöglicht. In kleinen und mittelständischen Unternehmen soll diese Anwendung eingesetzt werden, um deren Wirtschaftlichkeit zu verbessern. Diese Arbeit basiert daher auf einer Fallstudie in Zusammenarbeit mit der inovisio communications GmbH Chemnitz.

2. Theoretische Grundlagen

Im diesem Kapitel werden grundlegende und wichtige Kenngrößen und Methoden behandelt. Diese werden für die Berechnungen und die Lösungsansätze für die Anwendung in der Fallstudie benötigt.

2.1 Himmelsrichtung

Um die Bewegungsrichtung zwischen zwei Orten zu bestimmen, führte man die Himmelsrichtungen ein. Es gibt vier Haupthimmelsrichtungen (auch Kardinalpunkte genannt). Dies sind die Grundrichtungen Norden, Osten, Süden, Westen. Die Bewegungsrichtung gibt dabei an, in welche Himmelsrichtung man sich von einem Punkt A aus bewegt, um zu Punkt B zu gelangen. [1]

2.2 Geographische Koordinaten

Das Modell unseres Planeten Erde stellt man als Kugel dar. In Wirklichkeit ist die Erde keine Kugel, aber für die Darstellung ist es besser. Um die Position eines Ortes in diesem Kugelmodell definieren zu können, werden Koordinaten, geographische Länge (auch Longitude (**lon**) oder Längengrad genannt) und geographische Breite (auch als Latitude (**lat**) oder Breitengrad bezeichnet) verwendet. Diese Koordinaten sind Kugelkoordinaten. Die geographische Länge wird wie folgt definiert:

> „*Alle Meridiane verlaufen in Nord-Süd-Richtung und werden im „Grad westlicher Länge“ bzw. „Grad östlicher Länge“ angeben. Sie liegen am Äquator am weitesten auseinander. Zu den Polen hin laufen sie aufeinander zu.*“ [2]

und die geographische Breite:

> „*Alle Breitenkreise verlaufen in Ost-West-Richtung und werden in „Grad nördlicher Breite“ bzw. „Grad südlicher Breite“ angegeben. Die Breitenkreise haben zueinander (fast) gleiche Abstände. Sie schneiden Meridiane im rechten Winkel.*“ [2]

2.3 Darstellungsformen EPSG

Wie schon erwähnt kann ein Ort geographisch durch seine Breiten- und Längenangabe beschrieben werden. Dieses System ist ein internationales gebräuchliches Koordinatensystem mit der Bezeichnung **World Geodetic System 1984** (WGS84). WGS84 ist das einzige System, welches für jeden Kartendienst standardisiert und zwingend vorgeschrieben ist.
Das World Geodetic System 1984 ist nur eins von über 2500 Koordinatensystemen, die in Wissenschaft und Praxis Anwendung finden. 1986 entstand eine Arbeitsgruppe von europäischen Öl- und Gasfirmen, die **European Petroleum Survey Group** (EPSG). Die Gruppe erstellte eine Datenbank, in der jedem Koordinatensystem ein individueller EPSG-Code (auch *Spatial Reference Identifier* (SRID)) zugeordnet wurde. Dem System WGS84 teilte die Arbeitsgruppe den Code 4326 zu.
EPSG:900913 ist ein von Google, Bing, Yahoo und OpenStreetMap genutztes System (weiter auch „Google-Projektion“ bezeichnet). Heute ist der 900913-Code ein Synonym für die offizielle Bezeichnung als EPSG:3875. Der Grund für die Nutzung der Google-Projektion wird im Kapiel 2.4 „Transformation“ erklärt.

2.4 Transformation

Für die Nutzung von EPSG:4326 und EPSG:900913 müssen die Koordinaten transformiert werden, weil die EPSG:4326 nicht den Koordinaten von EPSG:900913 entsprechen. EPSG:4326 arbeitet mit den Koordinaten von geographischer Länge und Breite und EPSG:900913 nutzt kartesische Koordinaten mit x- und y-Werten.

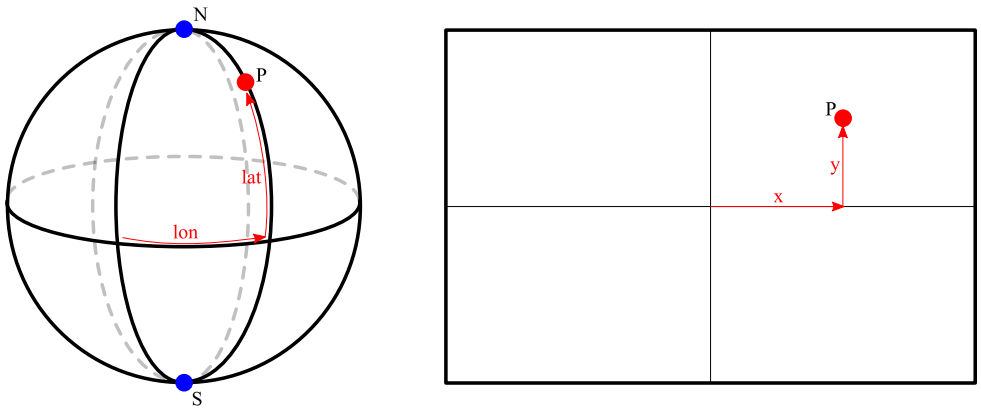

Abbildung 2.1: Darstellung von EPSG:4326 und EPSG:900913

Wie schon erwähnt, wird die Erde in Kugelform (dreidimensionale Darstellung (3D)) und Karten von Google und OpenLayers als flache Ebene (zweidimensional Darstellung (2D)) dargestellt. Die Abbildung von der gekrümmten Erdoberfläche auf eine Fläche kann ohne eine Projektion nicht stattfinden. Sonst würden große Verzerrungen zustande kommen. Die Transformation von Kugelkoordinaten (EPSG:4326) zu kartesischen Koordinaten (EPSG:900913) wird als ***Spherical Mercator*** bezeichnet. [3]

2.5 Problem Travelling Salesman (TSP)

Beim TSP besucht man bei einer Rundreise (Tour oder Route) alle n Orte $(A_1, A_2, ..., A_n)$ genau einmal und kehrt wieder zum Anfangspunkt zurück.
Zwischen je zwei Orten A und B gibt es eine Strecke mit den Kosten $c(A, B)$. Dies kann in einem Graphen $G = (V, E)$ veranschaulicht werden, indem die Orte als Knoten V, die Stecken als Kanten E und die Kosten $c\colon E \to \mathbb{N}$ zwischen zwei Orten A und B dargestellt werden.
Eine Rundreise $(A_1, A_2, ..., A_n, A_1)$ ist durch eine Permutation $\pi\colon \{1, 2, ..., n\} \to \{1, 2, ..., n\}$ der n Orte mit $\pi(j) = A_j, j = 1, ..., n$ gegeben. Für $c\colon \{A_1, A_2, ..., A_n\}^2 \to \mathbb{N}$ ergibt sich die Gesamtlänge π der Route wie folgt, wenn $c(A, B) = c(B, A)$ gilt (*symmetrisches* TSP):

$$\sum_{a=1}^{n-1} c(\pi(a), \pi(a+1)) + c(\pi(n), \pi(1))$$

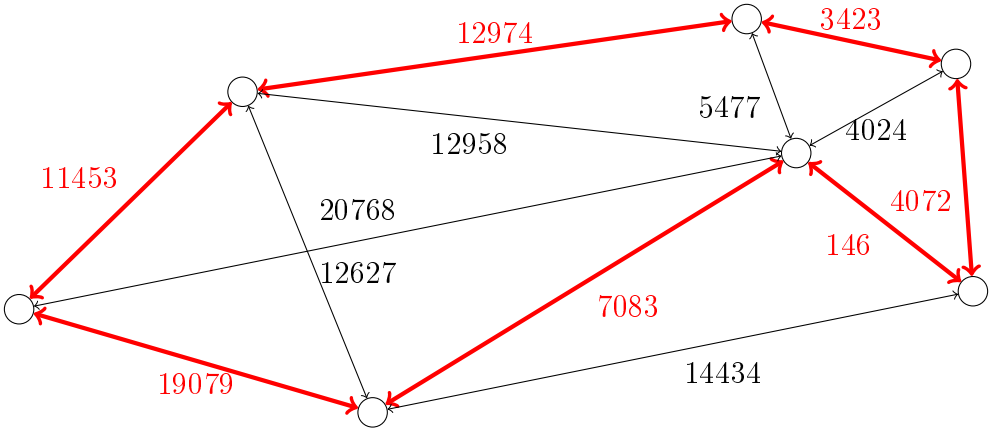

Abbildung 2.2: Beispiel für symmetrisches TSP

Aufgrund von Einbahnstraßen, Kreisverkehr, usw. gilt, dass $c(A, B) \neq c(B, A)$ ist. Deswegen muss mit gerichteten Graphen $G = (V, E, c)$, wobei $c\colon E \to \mathbb{N}$ eine Gewichtsfunktion ist, weiter gearbeitet werden.

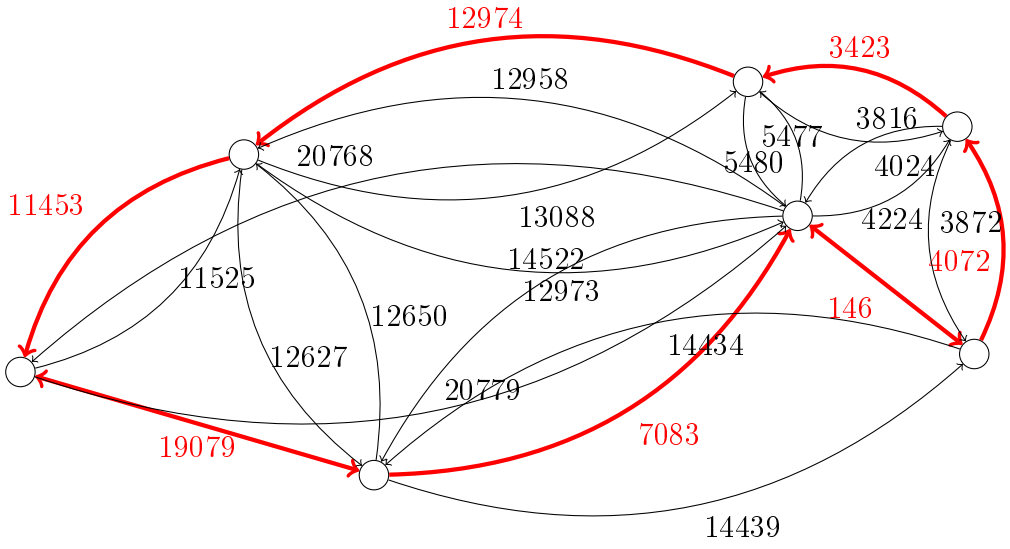

Abbildung 2.3: Beispiel für TSP mit gerichteten Graphen

Je nachdem, ob die kürzeste Tour (c = Streckenlänge in Metern) oder die schnellste Tour (c = Zeit in Sekunden) gesucht wird, muss darauf geachtet werden, welche Kosten c an die Funktion in der Webanwendung übergeben werden. [4]

2.6 Cluster

Bei den aufgezeichneten Koordinaten können sich Gruppen von Punkten bilden. Das Bilden von Gruppen kann aufgrund von Verkehr, Pausen oder Wartezeiten und durch Stehenbleiben an einer Stelle geschehen. Die Gruppen sind für die visuelle Ansicht in der Karte unübersichtlich und nicht erwünscht. Um die Ansicht übersichtlicher zu gestalten, wird das Verfahren „Clustering“ angewendet. Unter „Clustering“ („cluster“ engl. Gruppen, Haufen) oder „clustern“ versteht man das Vereinigen oder Zusammenführen von nah beieinander liegenden Punkten zu einem Punkt. Dieser Punkt repräsentiert als einziger alle folgenden Punkte für einen zuvor vorgegeben Umkreis. Somit kann eine für den Nutzer übersichtliche Anwendung präsentiert werden.

2.7 Mathematische Berechnungen

2.7.1 Entfernung

Um den Abstand zwischen zwei Punkten ($P_1 = \binom{P_{1_{lon}}}{P_{1_{lat}}}, P_2 = \binom{P_{2_{lon}}}{P_{2_{lat}}}$) zu ermitteln, gibt es zwei Möglichkeiten.

Einfache Berechnung (*Satz von Pythagoras*)
Mit dem *Satz des Pythagoras* erhält man nur eine Annäherung an die genaue Entfernung, weil die Erdkrümmung nicht beachtet wird. Diese Methode ist aber schneller in der Berechnung.

$$s_{einfach} = \sqrt{(P_{2_{lon}} - P_{1_{lon}})^2 + (P_{2_{lat}} - P_{1_{lat}})^2}$$

Genaue Berechnung (Bogenlänge)
Bevor mit der Berechnung begonnen werden kann, muss der Radius r oder der Durchmesser d der Erde bekannt sein. Der Durchmesser am Äquator beträgt 12756 Kilometer und von Nordpol zum Südpol 12714 Kilometer. Von diesen beiden Werten wird das arithmetische Mittel verwendet, welches 12735 Kilometer ergibt. [2]

$$\begin{aligned} d &= 12735 \\ r &= \frac{d}{2} = 6367,5 \end{aligned}$$

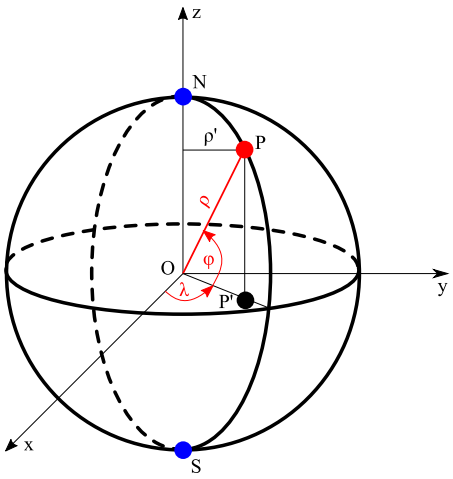

Abbildung 2.4: Kugelkoordinaten [2]

In der Abbildung wird die Erde als Kugel mit dem Mittelpunkt O im dreidimensionalen Raum veranschaulicht. Ein Ort kann als Punkt $P(\lambda, \varphi)$, wobei $\lambda = \text{lon}$ und $\varphi = \text{lat}$ ist, mit Kugelkoordinaten beschrieben werden und der Abstand ρ zum Mittelpunkt O entspricht dem Radius r. Somit kann dem Punkt P ein eindeutiges Tripel (ρ, λ, ϕ) zugeordnet werden.
Eine Transformation zwischen kartesischen Koordinaten (x, y, z) und Polarkoordinaten (ρ, λ, φ) kann mit $x = \rho \cdot \cos(\varphi) \cdot \cos(\lambda)$, $y = \rho \cdot \cos(\varphi) \cdot \sin(\lambda)$ und $z = \rho \cdot \sin(\varphi)$ vorgenommen werden.
Für die Berechnung des Winkels ϕ wird das Skalarprodukt $\overrightarrow{a} \cdot \overrightarrow{b} = a_x \cdot b_x + a_y \cdot b_y + a_z \cdot b_z$ zwischen zwei Vektoren $\overrightarrow{a} = (a_x, a_y, a_z)$ und $\overrightarrow{b} = (b_x, b_y, b_z)$ und die Beträge $|\overrightarrow{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2}$ und $|\overrightarrow{b}| = \sqrt{b_x^2 + b_y^2 + b_z^2}$ benötigt.
Der Winkel ϕ zwischen zwei Punkten auf dieser Kugel wird mit der folgenden Formel ermittelt:

$$
\begin{aligned}
\cos(\phi) &= \frac{\overrightarrow{p_1} \cdot \overrightarrow{p_2}}{|\overrightarrow{p_1}| \cdot |\overrightarrow{p_2}|} = \frac{\overrightarrow{p_1} \cdot \overrightarrow{p_2}}{\rho^2} \\
&= \frac{\begin{pmatrix} \rho \cdot \cos(\varphi_1) \cdot \cos(\lambda_1) \\ \rho \cdot \cos(\varphi_1) \cdot \sin(\lambda_1) \\ \rho \cdot \sin(\varphi_1) \end{pmatrix} \cdot \begin{pmatrix} \rho \cdot \cos(\varphi_2) \cdot \cos(\lambda_2) \\ \rho \cdot \cos(\varphi_2) \cdot \sin(\lambda_2) \\ \rho \cdot \sin(\varphi_2) \end{pmatrix}}{\rho^2} \\
&= \begin{pmatrix} \cos(\varphi_1) \cdot \cos(\lambda_1) \\ \cos(\varphi_1) \cdot \sin(\lambda_1) \\ \sin(\varphi_1) \end{pmatrix} \cdot \begin{pmatrix} \cos(\varphi_2) \cdot \cos(\lambda_2) \\ \cos(\varphi_2) \cdot \sin(\lambda_2) \\ \sin(\varphi_2) \end{pmatrix} \\
\phi &= \arccos(\cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \cos(\lambda_1 - \lambda_2) + \sin(\varphi_1) \cdot \sin(\varphi_2))
\end{aligned}
$$

Nach der Berechnung des Winkels ϕ, kann nun die Länge der Strecke s zwischen zwei Punkten berechnet werden. Zu beachten gilt, dass der Durchmesser und der Radius in Kilometer angegeben sind und in Meter umgerechnet werden müssen, damit die Berechnung korrekt erfolgt. [5] [6]

$$s = \frac{\phi \cdot \pi \cdot r}{180^\circ} \cdot 1000$$

2.7.2 Geschwindigkeit

Die Geschwindigkeit v zwischen zwei Orten A und B ist aussagekräftig, denn Personen und Objekte können sich auf der Erde nur mit einer bestimmten maximalen Geschwindigkeit bewegen. Das schnellste auf dem Boden bewegte öffentliche Verkehrsmittel in Deutschland ist der ICE 3 mit 330 $\frac{km}{h}$. Es wird im Rahmen dieser Arbeit und der Fallstudie festgelegt, dass kein Objekt und keine Person schneller als der ICE 3 ist. Die Übergänge zwischen Fahren (Pkw, Lkw, Bus, Bahn, etc.) und Laufen sind fließend. Die Kontrolle der Geschwindigkeit kann helfen, falsch gesetzte Punkte herauszufiltern. Gerechnet wird mit den physikalischen Basiseinheiten Meter und Sekunde. Die Geschwindigkeit berechnet sich aus der Entfernung s (in Meter) geteilt durch die Zeit t (in Sekunden) für eine zurückgelegte Strecke zwischen zwei Punkten A und B. [7]

$$v = \frac{s}{t}$$

Ein Meter pro Sekunde entspricht 3,6 Kilometern in der Stunde.

$$1\frac{\text{km}}{\text{h}} = \frac{1000 \text{ m}}{3600 \text{ sek}}$$
$$3,6\frac{\text{km}}{\text{h}} = 1\frac{\text{m}}{\text{s}}$$

2.7.3 Durchschnitt (arithmetisches Mittel)

Die Standortdaten der Basisstationen können in den Aufzeichnungen vorhanden sein, wegen fehlender GPS-Signale. Diese oder andere Fehler mit dem GPS sind meist durch plötzliche Sprünge im Routenverlauf in der Karte erkennbar. Um diese Orte wieder zu löschen nutzt man das arithmetische Mittel. Alle Längen der Teilstrecken werden addiert und durch die Anzahl dividiert.

$$\delta = \frac{\sum_{a=1}^{n-1} c(\pi(a), \pi(a+1))}{(n-1)}$$

Anschließend wird zu dem Durchschnitt δ ein Wert δ_{Tol} berechnet, der sich aus dem Produkt von δ und einem Toleranzwert μ ergibt, also

$$\delta_{Tol} = \delta \cdot \mu$$

Mit δ_{Tol} wird jede Strecke zwischen je zwei Orten A und B verglichen und beide Orte erhalten eine Markierung, ob die Route kleiner oder gleich δ_{Tol} ist. Auch die Durchschnittsgeschwindigkeit lässt sich mithilfe von δ ermitteln.

2.7.4 Determinante

Die Zahl $D = a_{11}a_{22} - a_{12}a_{22}$ heißt **Determinante** des Koeffizientenschemas $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ oder auch *Koeffizientendeterminante*. Man schreibt: $D = a_{11}a_{22} - a_{12}a_{22} = \det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ oder auch $D = a_{11}a_{22} - a_{12}a_{22} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$. [5]

2.7.5 Himmelsrichtung

Benötigt wird die Richtung $R(P_1 = \binom{P_{1_{lon}}}{P_{1_{lat}}}, P_2 = \binom{P_{2_{lon}}}{P_{2_{lat}}})$ zwischen zwei Punkten. Als Ausgabe soll „N“ für Norden, „NE“ für Nordost, „E“ für Ost, „SE“ für Südost, „S“ für Süden, „SW“ für Südwest, „W“ für West und „NW“ für Nordwest zurückgegeben werden.
Zuerst ist zu überlegen, welche Daten mit den zwei vorhandenen Punkten errechnet werden können. Zwischen P_1 und P_2 kann ein Vektor $\overrightarrow{v} = \binom{P_{2_{lon}} - P_{1_{lon}}}{P_{2_{lat}} - P_{1_{lat}}}$ aufgestellt werden. Um einen Winkel ϕ berechnen zu können, wird ein weiterer Vektor benötigt. Dazu kann ein Vektor mit Verlauf in Richtung Norden von Punkt P_1 erzeugt werden, also $\overrightarrow{v_N} = \binom{0}{1}$.

$$\begin{aligned}
\cos(\phi) &= \frac{\overrightarrow{v} \cdot \overrightarrow{v_N}}{|\overrightarrow{v}| \cdot |\overrightarrow{v_N}|} \\
&= \frac{v_x \cdot v_{N_x} + v_y \cdot v_{N_y}}{\sqrt{v_x^2 + v_y^2} \cdot \sqrt{v_{N_x}^2 + v_{N_y}^2}} \\
\phi &= \arccos\left(\frac{v_x \cdot v_{N_x} + v_y \cdot v_{N_y}}{\sqrt{v_x^2 + v_y^2} \cdot \sqrt{v_{N_x}^2 + v_{N_y}^2}}\right)
\end{aligned}$$

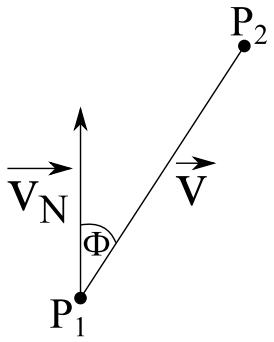

Abbildung 2.5: Vektor zur Bestimmung der Himmelsrichtung

Der Winkel ϕ wird nur im Bereich $0° \leq \phi \leq 180°$ angezeigt. Somit kann nur zwischen Norden und Süden unterschieden werden. Es fehlen Osten und Westen, um damit auch die restlichen vier Himmelsrichtungen zu bestimmen. Dabei verwendet man Determinanten. Für die Determinante werden zwei Vektoren, die bereits vorhanden sind, benötigt und setzt diese in die Funktion ein. Die Funktion lautet:

$$\det(\overrightarrow{v}, \overrightarrow{v_N}) = \overrightarrow{v_x} \cdot \overrightarrow{v_{N_y}} - \overrightarrow{v_y} \cdot \overrightarrow{v_{N_x}}$$

Angenommen, wir haben $\phi = 90°$ errechnet und das Ergebnis der Determinante aus den zwei Vektoren ist größer gleich Null, so ist das Osten. Andernfalls ergibt $\phi := 360° - \phi$ und es kann die Himmelsrichtung bestimmt werden:

$$\begin{cases} 22,5° \leq \phi < 67,5° & \text{NE} \\ 67,5° \leq \phi < 112,5° & \text{E} \\ 112,5° \leq \phi < 157,5° & \text{SE} \\ 157,5° \leq \phi < 202,5° & \text{S} \\ 202,5° \leq \phi < 247,5° & \text{SW} \\ 247,5° \leq \phi < 292,5° & \text{W} \\ 292,5° \leq \phi < 337,5° & \text{NW} \\ \text{sonst} & \text{N} \end{cases}$$

2.7.6 Zick-Zack

Ungewollte Zick-Zack-Bewegungen sollen geglättet werden. Um zu prüfen, ob so ein Verlauf in einem Teilstück der Route enthalten ist, benutzt man vier aufeinanderfolgende Punkte und die Funktion zur Richtungsbestimmung. Für jeden Abschnitt der Teilroute (P_1, P_2, P_3, P_4) vom ersten bis zum letzten Punkt müssen die Richtungen R, also $R_{\overrightarrow{1,2}}(P_1, P_2), R_{\overrightarrow{2,3}}(P_2, P_3), R_{\overrightarrow{3,4}}(P_3, P_4)$, ermittelt werden. Zum Schluss wird verglichen:

Zeilennummer	$R_{\overrightarrow{a,b}}$	$R_{\overrightarrow{b,c}}$
1	N	SE, S, SW
2	NE	S, SW, W
3	E	SW, W, NW
4	SE	W, NW, N
5	S	NW, N, NE
6	SW	N, NE, E
7	W	NE, E, SE
8	NW	E, SE, S

Tabelle 2.1: Vergleichstabelle zur Bestimmung von Zick-Zack-Bewegungen

Falls $i, j \in \{1, 2, 3, 4, 5, 6, 7, 8\}$ existieren, so dass $R_{\overrightarrow{2,3}}$ der Eintrag $R_{\overrightarrow{a,b}}$ in der Zeile i in der Tabelle ist und $R_{\overrightarrow{3,4}}$ in der Menge $R_{\overrightarrow{b,c}}$ in der Zeile i enthalten ist und gleichzeitig $R_{\overrightarrow{1,2}}$ der Eintrag $R_{\overrightarrow{a,b}}$ in der Zeile j in der Tabelle ist und $R_{\overrightarrow{2,3}}$ in der Zeile j in der Menge $R_{\overrightarrow{b,c}}$ enthalten ist, dann ist ein Zick-Zack-Verlauf in der Route vorhanden. Trifft dieser Fall ein, so wird der Punkt P_3 gelöscht.

2.7.7 Schleife

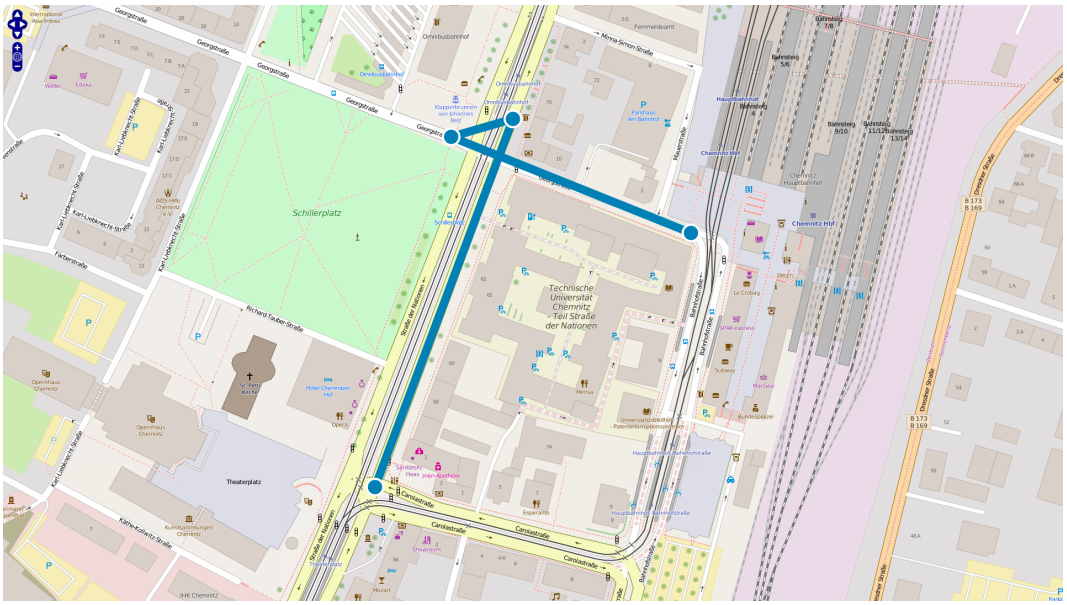

Abbildung 2.6: Beispiel für eine Schleife in der Route

Eine Schleife, bei der $\overline{P_1P_2}$ und $\overline{P_3P_4}$ lange Strecken sind und $\overrightarrow{P_2P_3}$ kurz ist, soll korrigiert werden (siehe Abbildung 2.6). Solch eine Schleife kann durch Messfehler des GPS-Signals im Gerät entstehen. Falls die drei Strecken ungefähr die gleiche Länge haben, wird keine Korrektur vorgenommen.
Um nun dieses Problem der Schleife zu lösen, werden immer die letzten vier Punkte beim Einzeichnen in die Karte betrachtet. Geprüft wird, ob sich die Strecken $\overline{P_1P_2}$ und $\overline{P_3P_4}$ schneiden.

$$\overline{P_aP_b} = \begin{pmatrix} P_{a_{lon}} \\ P_{a_{lat}} \end{pmatrix} + x \begin{pmatrix} (P_{b_{lon}} - P_{a_{lon}}) \\ (P_{b_{lat}} - P_{a_{lat}}) \end{pmatrix}$$

$$\overline{P_1P_2} = \begin{pmatrix} P_{1_{lon}} \\ P_{1_{lat}} \end{pmatrix} + x_1 \begin{pmatrix} (P_{2_{lon}} - P_{1_{lon}}) \\ (P_{2_{lat}} - P_{1_{lat}}) \end{pmatrix} \qquad \overline{P_3P_4} = \begin{pmatrix} P_{3_{lon}} \\ P_{3_{lat}} \end{pmatrix} + x_2 \begin{pmatrix} (P_{4_{lon}} - P_{3_{lon}}) \\ (P_{4_{lat}} - P_{3_{lat}}) \end{pmatrix}$$

Unter der Voraussetzung, dass die zwei Strecken nicht parallel zueinander oder aufeinander verlaufen, gilt:

$$\overline{P_1P_2} = \overline{P_3P_4} \quad \Leftrightarrow \quad \begin{pmatrix} P_{1_{lon}} \\ P_{1_{lat}} \end{pmatrix} + x_1 \begin{pmatrix} (P_{2_{lon}} - P_{1_{lon}}) \\ (P_{2_{lat}} - P_{1_{lat}}) \end{pmatrix} = \begin{pmatrix} P_{3_{lon}} \\ P_{3_{lat}} \end{pmatrix} + x_2 \begin{pmatrix} (P_{4_{lon}} - P_{3_{lon}}) \\ (P_{4_{lat}} - P_{3_{lat}}) \end{pmatrix}$$

$$\text{I)} \quad P_{1_{lon}} + x_1(P_{2_{lon}} - P_{1_{lon}}) = P_{3_{lon}} + x_2(P_{4_{lon}} - P_{3_{lon}})$$
$$\text{II)} \quad P_{1_{lat}} + x_1(P_{2_{lat}} - P_{1_{lat}}) = P_{3_{lat}} + x_2(P_{4_{lat}} - P_{3_{lat}})$$

Nach x_1 auflösen

$$\begin{aligned} P_{1_{lon}} + x_1(P_{2_{lon}} - P_{1_{lon}}) &= P_{3_{lon}} + x_2(P_{4_{lon}} - P_{3_{lon}}) \\ x_1(P_{2_{lon}} - P_{1_{lon}}) &= P_{3_{lon}} + x_2(P_{4_{lon}} - P_{3_{lon}}) - P_{1_{lon}} \\ x_1 &= \frac{P_{3_{lon}} + x_2(P_{4_{lon}} - P_{3_{lon}}) - P_{1_{lon}}}{(P_{2_{lon}} - P_{1_{lon}})} \end{aligned}$$

x_1 in II einsetzen und nach x_2 auflösen

$$\begin{aligned} P_{1_{lat}} + x_1(P_{2_{lat}} - P_{1_{lat}}) &= P_{3_{lat}} + x_2(P_{4_{lat}} - P_{3_{lat}}) \\ x_2(P_{4_{lat}} - P_{3_{lat}}) &= P_{1_{lat}} + x_1(P_{2_{lat}} - P_{1_{lat}}) - P_{3_{lat}} \\ x_2 &= \frac{P_{1_{lat}} + x_1(P_{2_{lat}} - P_{1_{lat}}) - P_{3_{lat}}}{(P_{4_{lat}} - P_{3_{lat}})} \end{aligned}$$

Wenn der Schnittpunkt von $\overline{P_1P_2}$ und $\overline{P_3P_4}$ zwischen den Punkten P_3 und P_4 liegt ($0 \leq x_2 \leq 1$), also auf der Strecke $\overline{P_3P_4}$, dann liegt eine Schleife wie im Bild vor. Dieser Fall wird dann gelöst, in dem der Punkt P_2 gelöscht wird.
Man könnte auch die Punkte P_2 und P_3 entfernen und den Schnittpunkt einfügen. Damit wird eine bessere graphische Darstellung ermöglicht, aber ein nicht originaler Punkt genutzt. Dies ist im Rahmen der Fallstudie mit dem Praxisunternehmen jedoch nicht erwünscht, da dies die aufgezeichneten Daten manipuliert bzw. verändert.
Die Vorgehensweise mit dem Lösen der Gleichung ist zwar mathematisch gesehen korrekt, aber anfällig für Fehler bei der Programmierung. Da auch der genaue Schnittpunkt nicht benötigt wird, kann der Versuch mit den bereits existierenden Funktionen gestartet werden, diesen Fall zu lösen. Wenn sich die Strecken $\overline{P_1P_2}$ und $\overline{P_3P_4}$ schneiden würden, so bilden die Punkte P_2, P_3 und der Schnittpunkt S von den Strahlen $\overline{P_1P_2}$ und $\overline{P_3P_4}$ ein Dreieck. Für jedes Dreieck gilt der Innenwinkelsatz:

$$180° = \alpha + \beta + \gamma \quad \text{für } \alpha, \beta, \gamma \geq 0$$

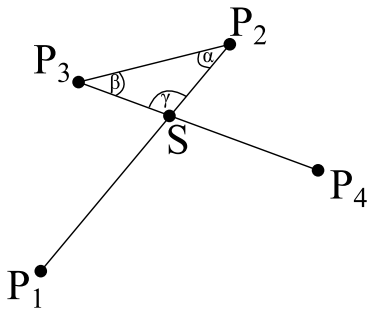

Abbildung 2.7: Schleife - graphische Darstellung für die Berechnung

Mit der Winkelfunktion können die Winkel α mit den Punkten P_1, P_2 und P_3 und β mit den Punkten P_2, P_3, sowie P_4, wie im Abschnitt 2.7.1 beschrieben bestimmt werden. Daraus folgt, dass der Winkel $\gamma = 180° - \alpha - \beta$ groß ist. Nun muss die Länge der Strecke zwischen dem Punkt P_3 und dem Schnittpunkt S ermittelt werden. Wenn die Strecke $\overline{P_3P_4}$ kleiner als die Strecke $\overline{P_3S}$ ist, so schneiden sich die Strecken $\overline{P_1P_2}$ und $\overline{P_3P_4}$ nicht. Sonst schneiden sich die zwei Strecken und der Punkt P_3 muss gelöscht werden. Die einzige nützliche Strecke ist die Strecke $\overline{P_2P_3}$, da diese Länge als einzige im Dreieck P_2P_3S bekannt ist. Der Sinussatz kann zur Berechnung der Streckenlänge $\overline{P_3S}$ verhelfen:

$$\frac{\overline{P_3S}}{\sin(\alpha)} = \frac{\overline{SP_2}}{\sin(\beta)} = \frac{\overline{P_2P_3}}{\sin(\gamma)}$$

Also, gegeben sind alle drei Winkel α, β und γ, sowie die Strecke $\overline{P_2P_3}$. Gesucht wird nun die Länge der Strecke $\overline{P_3S}$. Berechnet wird diese wie folgt:

$$\overline{P_3S} = \frac{\overline{P_2P_3}}{\sin(\gamma)} \cdot \sin(\alpha)$$

Wenn $\overline{P_3P_4}$ kleiner als $\overline{P_3S}$ ist, so schneiden sich $\overline{P_1P_2}$ und $\overline{P_3P_4}$ nicht. Andernfalls wird der Punkt P_3 gelöscht.

2.7.8 Haiflosse (Shark Fin)

Neben den Schleifen und Zick-Zack-Bewegungen können auch „Haiflossen“ (auch Shark Fin genannt) im Verlauf einer Route entstehen. Diese sind plötzlich auftauchende Ecken auf geraden Strecken, die möglicherweise durch die Übermittlung falscher GPS-Koordinaten, das Löschen oder das Clustern vorheriger Punkte auftauchen. Solche Ecken sollen ebenfalls aus dem Verlauf

gelöscht werden.

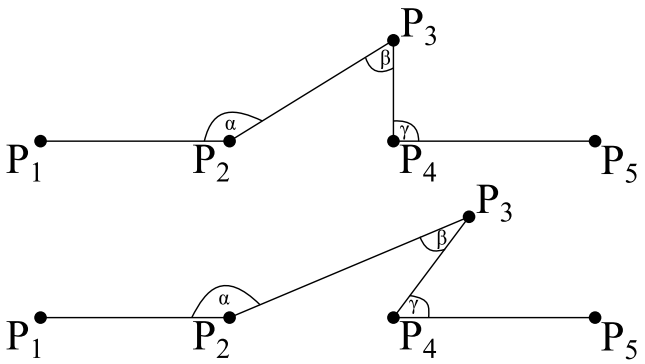

Abbildung 2.8: Haiflosse - graphische Darstellungen für mögliche Verläufe

Die Erkennung einer solchen „Haiflosse“ ist so definiert, dass jeweils für fünf im Verlauf aufeinander folgende Punkte in der Gesamtroute geprüft wird, ob entweder $\alpha, \beta > 90°$ und $\gamma \leq 90°$ oder $\alpha \leq 90°$ und $\beta, \gamma > 90°$ oder $\alpha, \gamma > 90°$ und $\beta \leq 160°$ gilt und gleichzeitig die Winkel $\sphericalangle(P_1, P_2, P_4)$, $\sphericalangle(P_2, P_4, P_5) > 160°$ sind. Tritt einer dieser Fälle ein, so wird der Punkt P_3 gelöscht, sonst nicht.

2.7.9 Gelöschte Punkte in der Nähe vom Routenverlauf

Durch das Auswerten mit den oben genannten Funktionen der aufgezeichneten Punkte kommt es zum Löschen von Punkten, die aber zum Verlauf der Route gehören. Deswegen werden zwischen zwei Punkten P_1 und P_2, die auf jeden Fall in die Karte gezeichnet werden, die dazwischen liegenden Punkte, die bereits aussortiert wurden, noch einmal überprüft. Dafür gibt es zwei Varianten.
Die erste Variante ist den Abstand h_{Lot} (das Lot) zwischen der Strecke $\overline{P_1P_2}$ und dem Punkt P_P zu prüfen. Es wird eine maximalen Höhe h_{max} zwischen dem Punkt P_P und der Strecke $\overline{P_1P_2}$ festgelegt. Ist das Lot kleiner oder gleich der maximalen Höhe h_{max}, dann wird der Punkt wieder hinzugefügt, ansonsten nicht. Der Abstand h_{Lot} zwischen P_P und $\overline{P_1P_2}$ kann mit zwei Methoden bestimmt werden.

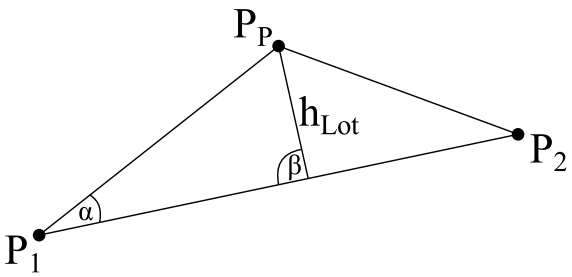

Abbildung 2.9: Lot in einem Dreieck

In der ersten Methode nutzt man den Flächeninhalt. Der Flächeninhalt A lässt sich so ermitteln:

$$A = |\frac{1}{2} \cdot \det(\overline{P_1P_2}, \overline{P_2P_P})| = \frac{1}{2} \cdot \overline{P_1P_2} \cdot h_{Lot}$$

Umgestellt ergibt sich dann das Lot:

$$h_{Lot} = \frac{|\det(\overline{P_1P_2}, \overline{P_2P_P})|}{\overline{P_1P_2}}$$

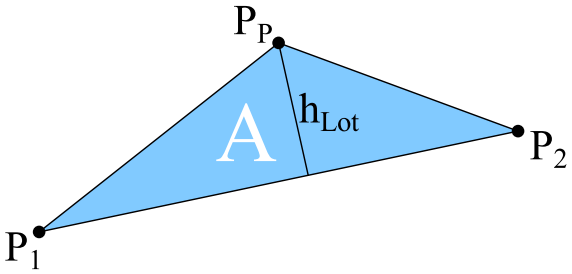

Abbildung 2.10: Lot berechnen durch Flächeninhalt

Die zweite Methode ist von dem Abschnitt 2.7.7 bekannt, unter Anwendung des Sinussatzes:

$$\begin{aligned} h_{Lot} &= \frac{\overline{P_1P_P}}{\sin(\beta)} \cdot \sin(\alpha) \qquad \text{für } \beta = 90°;\ \alpha = \arccos\left(\frac{\overline{P_1P_2} \cdot \overline{P_1P_P}}{|\overline{P_1P_2}| \cdot |\overline{P_1P_N}|}\right) \\ &= \overline{P_1P_P} \cdot \sin(\alpha) \end{aligned}$$

Für beide Methoden gilt jedoch, wenn h_{Lot} größer als h_{max} ist, dann wird der

Punkt nicht eingefügt, sonst wird er wieder eingefügt.

Diese Variante ist möglich, aber bringt unerwünschte Seiteneffekte mit sich. Je näher die Punkte P_1 und P_2 beieinander liegen, umso höher ist die Wahrscheinlichkeit, dass eine Haiflosse (auch „Shark Fin“ bezeichnet) entsteht, die nicht erwünscht ist. Wenn sich P_1 und P_2 nähern können die Winkelgrößen wie in den Abschnitt 2.7.8 beschrieben besitzen. Also wird diese Möglichkeit nicht verwendet.

Die zweite Möglichkeit ist einfach und verhindert auch nicht gewollte „Haiflossen“ zu erzeugen. Dafür wird die Winkelfunktion wieder eingesetzt:

$$\begin{aligned} \cos(\gamma) &= \frac{\overrightarrow{P_P P_1} \cdot \overrightarrow{P_P P_2}}{|\overrightarrow{P_P P_1}| \cdot |\overrightarrow{P_P P_2}|} \\ \gamma &= \arccos\left(\frac{\overrightarrow{P_P P_1} \cdot \overrightarrow{P_P P_2}}{|\overrightarrow{P_P P_1}| \cdot |\overrightarrow{P_P P_2}|}\right) \end{aligned}$$

Wenn der Winkel zwischen P_1, P_P und P_2 ($\sphericalangle(P_1 P_P P_2)$) größer als 160° (siehe Abschnitt 2.7.8) ist, so wird der Punkt P_P wieder in die Route aufgenommen, sonst nicht.

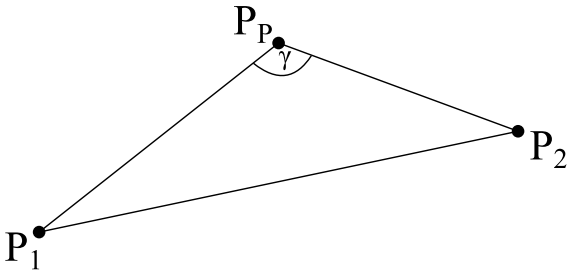

Abbildung 2.11: Winkel γ bestimmen

3. Fallstudie

3.1 Zielsetzung

Ziel dieser Fallstudie ist die Entwicklung einer Webanwendung zur Planung von Routen und nach Beendigung der Reise eine visuelle Betrachtung der tatsächlich zurückgelegten Strecke für den Nutzer. Nachfolgend wird das Vorgehen im Rahmen dieser Fallstudie beschrieben.

3.2 Unternehmen und Kunden

Es gibt bereits Anwendungen, mit denen Routen geplant und verfolgt werden können. Diese zielen insbesondere auf große Unternehmen ab 250 bis 500 Mitarbeiter ab. Kleine und mittelständische Unternehmen (KMU) können solche Systeme oft aus Kostengründen nicht in ihre Wertschöpfungsprozesse implementieren. Mittels der Webanwendung soll genau diese Nische im KMU-Segment geschlossen werden.
Am Beispiel eines Unternehmens mit einem Hauptsitz und neun Filialen wird diese Fallstudie umgesetzt.

Standort	**lon**	**lat**
Hauptsitz (H)	13.276266	51.280717
Filiale 1 (F1)	13.278189	51.280110
Filiale 2 (F2)	13.123148	51.123317
Filiale 3 (F3)	13.101218	51.121286
Filiale 4 (F4)	13.113559	51.298723
Filiale 5 (F5)	13.151779	51.204518
Filiale 6 (F6)	13.293214	51.306915
Filiale 7 (F7)	13.261708	51.320098
Filiale 8 (F8)	13.047889	51.234822
Filiale 9 (F9)	13.213819	51.242446

Tabelle 3.1: Standorte eines Unternehmens [8]

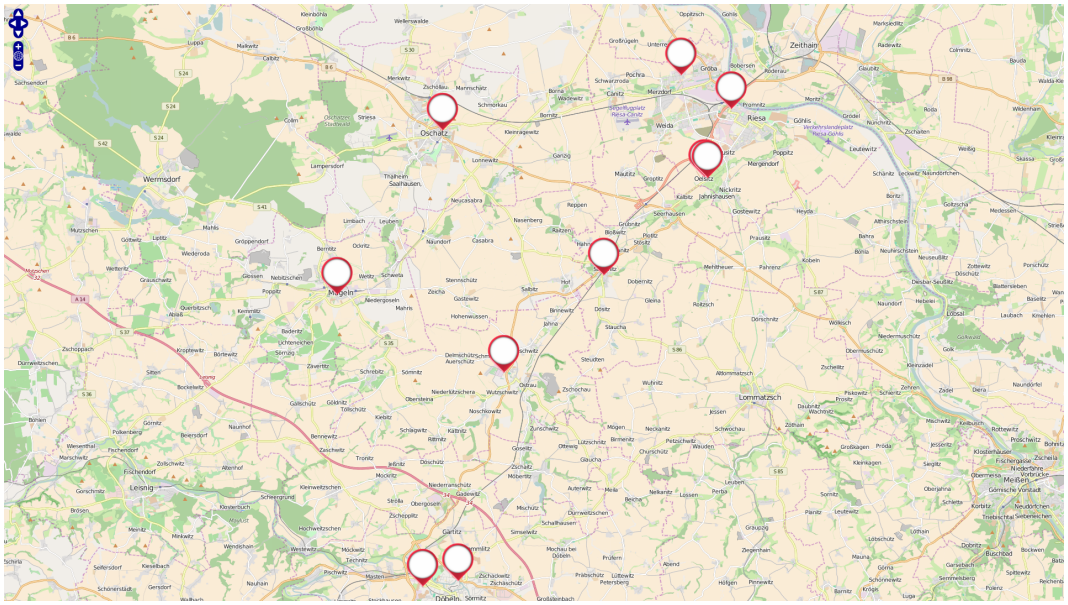

Abbildung 3.1: Übersicht mit allen Standorten [8]

Es sollen alle Filialen genau einmal angefahren werden. Der Start und das Ziel ist der Hauptsitz. Die kürzeste Route nach Kilometern soll unter Einhaltung der Straßenverkehrsordnung (StVO), unter Beachtung vorhandener Umleitungen bei Baustellen und unter Berücksichtigung der Verkehrslage befahren werden.

3.3 Umsetzung

3.3.1 Vorbereitung und Schaffung technischer Grundlagen

OpenLayers

Für die Webanwendung wird das Kartenmaterial von OpenStreetMap und die JavaScript-Bibliothek OpenLayers (Version 2.13.1) genutzt. Benötigt wird ein HTML-Dokument, das den JavaScript-Code ausführt und auf dem Monitor darstellt. Auch die JavaScript-Bibliothek jQuery (Version 3.1.0) wird in der Anwendung genutzt. Im HTML-Dokument werden die benötigten JavaScript-Bibliotheken geladen.

Beim Öffnen der HTML-Seite soll eine Karte angezeigt werden, die auf einen Ort zentriert ist (zum Beispiel die Technische Universität Chemnitz). Um dies zu erlangen, muss im Platzhalter für das JavaScript eine Variable „var map“ für die Karte angelegt werden, die als OpenLayers (OL) Map definiert wird. Die Karte „map“ besitzt zudem eine Navigations- und Zoomleiste und wird

einem HTML-Element mit derselben ID zugeordnet. Anschließend wird das Kartenlayer „Mapnik“ von der OpenStreetMap-Bibliothek (OSM) zur Karte hinzugefügt. Damit die Karte auch im HTML-Dokument angezeigt wird, muss in dem JavaScript die Karte „map“ auf einen Punkt (mit Transformation von EPSG:4326 zu ESPG:900913, wie in Kapitel 2.3 beschrieben) und ein Zoomlevel ausgerichtet werden und im HTML-Körper (weiter auch als „body“ genannt oder „<body>“ dargestellt) muss ein HTML-Element eingefügt werden mit der selben ID wie bei „map“ definiert. [3] [9]

```
1  <!DOCTYPE html>
2
3  <html style="height: 100%; margin: 0px">
4      <head>
5          <meta charset="utf-8">
6          <meta http-equiv="Content-Language" content="de">
7          <title>Implementierung einer webbasierten Anwendung zur Routenplanung
               und -optimierung anhand einer Fallstudie - Karte</title>
8          <script type="text/javascript"
               src="https://code.jquery.com/jquery-3.1.0.js"></script>
9          <script type="text/javascript"
               src="http://www.openlayers.org/api/OpenLayers.js"></script>
10         <script type="text/javascript"
               src="http://www.openstreetmap.org/openlayers/OpenStreetMap.js">
               </script>
11         <script type="text/javascript">
12             $(document).ready(function() {
13                 var fromProjection = new OpenLayers.Projection("EPSG:4326");
14                 var toProjection = new OpenLayers.Projection("EPSG:900913");
15
16                 var map = new OpenLayers.Map('map', {
17                     controls: [
18                         new OpenLayers.Control.PanZoom(),
19                         new OpenLayers.Control.Navigation()
20                     ]
21                 });
22
23                 map.addLayer(
24                     new OpenLayers.Layer.OSM.Mapnik("Mapnik", {
25                         transitionEffect: 'resize'
26                     })
27                 );
28
29                 map.setCenter(new OpenLayers.LonLat(12.928078,
                       50.839021).transform(fromProjection, toProjection), 18);
30             });
31         </script>
32     <head/>
33
34     <body style="height: 100%; margin: 0px">
35         <div id="map" style="height: 100%; margin: 0px">
36         </div>
37     </body>
38 </html>
```

Abbildung 3.2: Karte

Bestimmte Orte sollen mit einem Punkt oder Icon gekennzeichnet werden. Dafür wird eine neue Ebene (auch als „Layer“ bezeichnet) in die Karte eingefügt. Dem Layer gibt man einen Namen und legt die Füllfarbe, die Randfarbe, den Radius, die Linienstärke des Rahmens und die Form der Ecken der Rahmenlinie als Standarddesign fest. Zum Schluss fügt man den Punkt mit den entsprechenden Koordinaten in das Layer „POI“ ein. [3] [9]

```
1  <!DOCTYPE html>
2
3  <html style="height: 100%; margin: 0px">
4      <head>
5          <meta charset="utf-8">
6          <meta http-equiv="Content-Language" content="de">
7          <title>Implementierung einer webbasierten Anwendung zur Routenplanung
               und -optimierung anhand einer Fallstudie - Karte</title>
8          <script type="text/javascript"
               src="https://code.jquery.com/jquery-3.1.0.js"></script>
9          <script type="text/javascript"
               src="http://www.openlayers.org/api/OpenLayers.js"></script>
10         <script type="text/javascript"
               src="http://www.openstreetmap.org/openlayers/OpenStreetMap.js">
               </script>
11         <script type="text/javascript">
12             $(document).ready(function() {
13                 var fromProjection = new OpenLayers.Projection("EPSG:4326");
14                 var toProjection = new OpenLayers.Projection("EPSG:900913");
15
16                 var map = new OpenLayers.Map('map', {
17                     controls: [
18                         new OpenLayers.Control.PanZoom(),
19                         new OpenLayers.Control.Navigation()
20                     ]
```

```
21                  });
22
23                  map.addLayer(
24                      new OpenLayers.Layer.OSM.Mapnik("Mapnik", {
25                          transitionEffect: 'resize'
26                      })
27                  );
28
29                  map.addLayer(
30                      new OpenLayers.Layer.Vector("POI", {
31                          styleMap: new OpenLayers.StyleMap({
32                              'default': new OpenLayers.Style({
33                                  'fillColor': 'transparent',
34                                  'pointRadius': 10,
35                                  'strokeWidth': 5,
36                                  'strokeColor': '#ff0000',
37                                  'strokeLinecap': 'round'
38                              })
39                          })
40                      })
41                  );
42
43                  map.layers[1].addFeatures(new OpenLayers.Feature.Vector(new
                        OpenLayers.Geometry.Point(12.928078,
                        50.839021).transform(fromProjection, toProjection)));
44
45                  map.setCenter(new OpenLayers.LonLat(12.928078,
                        50.839021).transform(fromProjection, toProjection), 18);
46              });
47          </script>
48      <head/>
49
50      <body style="height: 100%; margin: 0px">
51          <div id="map" style="height: 100%; margin: 0px">
52          </div>
53      </body>
54  </html>
```

Abbildung 3.3: Karte mit Markierung

Geo-Daten und Geo-Server

Wie im Abschnitt 2.2 wird ein Ort mittels geographischer Koordinaten (EPSG:4326) beschrieben. Wenn diese Koordinaten bekannt sind, dann können diese verwendet werden. Orte in Städten oder Dörfern sind mit Adressen (Straßenname, Hausnummer, Postleitzahl und Stadt- bzw. Dorfnamen) beschrieben. Straßen außerhalb von bewohnten Gebieten besitzen oft keine Straßennamen, sondern nur einen Buchstaben für die Art der Straße und eine Indexzahl. Wälder, Flüsse, Seen, Berge und Gebirge sind durch ihre Namen zu unterscheiden. Weitere Merkmale, um einen Ort zu bestimmen, sind zum Beispiel die kontinentale Lage und Staatszugehörigkeit.

Um von einem Adressatz die geographischen Koordinaten zu erhalten, benötigt man einen Geo-Server. Der Geo-Server enthält eine Datenbank mit Adressen, Wälder, Flüssen, usw. und deren geografischen Angaben (kurz Geo-Daten). Ein Objekt (Gebäude, See, Berg, usw.) besitzt theoretisch und praktisch mehrere geografische Koordinaten, wird eventuell aber als nur ein Punkt dargestellt. Der Grund dafür ist die Vermeidung von Redundanz in den Datenbeständen. Jedes Objekt wird durch einen Punkt repräsentiert (Prinzip wie beim Cluster - Abschnitt 2.6). Bekannte Geo-Server sind Google Maps, Bing Maps, HERE und AppleMaps.

Stellt man eine Anfrage an den Server mit einem Adressatz, so erhält man von diesem Adressatz, sofern er existiert, die Geo-Daten. Auch kann an den Server eine geografische Länge und Breite übergeben werden. Als Antwort bekommt man ein oder mehrere nah liegende Objekte mit den dazugehörigen Geo-Daten zurück.

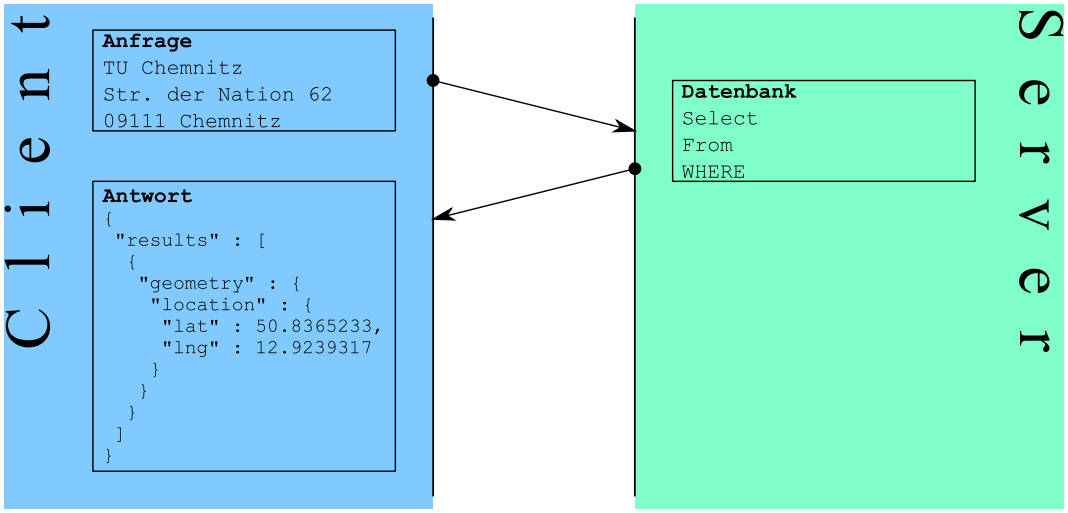

Abbildung 3.4: Geo-Server - Anfrage mit Adressatz [10]

Der Geo-Server kann auch eine Route zwischen zwei Orten berechnen. Dafür wird von jedem Ort die geographische Länge und Breite benötigt. Anschließend berechnet er mit einem auf dem Server befindlichen Algorithmus die Route und antwortet mit dem Routenverlauf und der Routenbeschreibung.

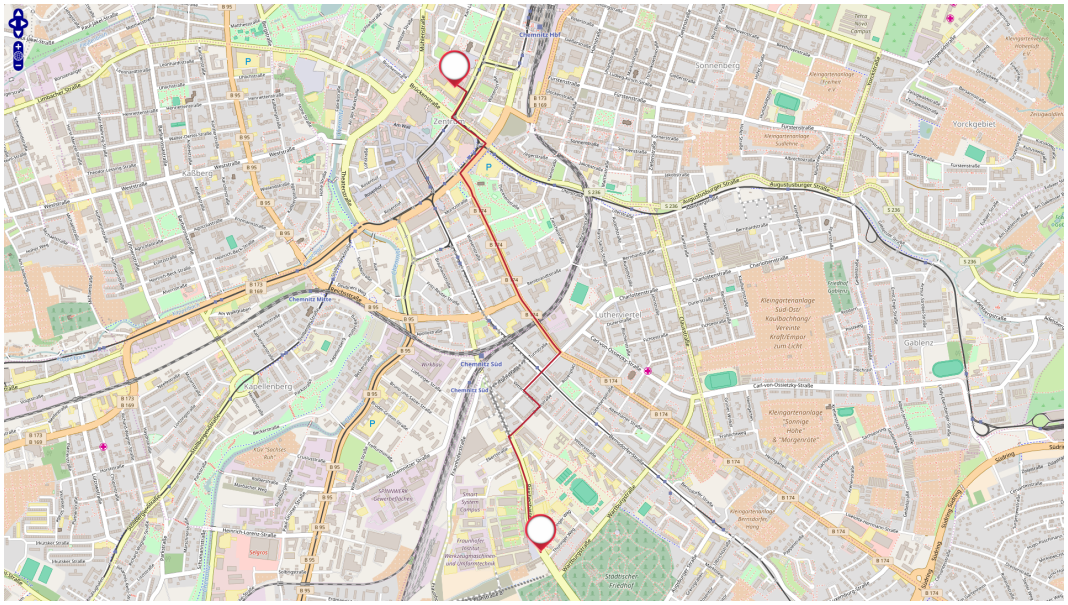

Abbildung 3.5: Geo-Server - Route berechnet [10]

Adjazenzmatrix

Für die spätere Berechnung werden zwischen den einzelnen Orten die Entfernungen benötigt. An den Geo-Server müssten dann n^2 (n = Start + i Zwischenziele + Ziel, $n \in \mathbb{N}$) Anfragen gestellt werden. Die Anfrage von einem Ort zu sich selbst wird nicht an den Geo-Server gestellt, da die Entfernung stets Null (Meter oder Sekunden) ist. Damit spart man n Anfra-

gen an den Server. Des weiteren können die Entfernungen zwischen allen Zwischenzielen zum Start und vom Ziel zum Start mit unendlich (in der Programmierung mit Null) definieren und somit weitere $n-1$ Anfragen zum Geo-Server sparen. So gelangt man nicht vom Ziel über die Zwischenstopps zum Start, sondern vom Start über die Zwischenziele zum Ziel. Somit können auch die Entfernungen vom Ziel zu den Zwischenzielen mit unendlich (in der Programmierung mit Null) gesetzt werden, da man von den Zwischenstopps zum Ziel möchte und nicht umgekehrt. Es kann wieder an Anfragen gespart werden. Dieses Mal $n-2$ Anfragen, die nicht an den Geo-Server getätigt werden müssen. Am Ende sind $n^2 - n - (n-1) - (n-2) = n^2 - 3n + 3$ Anfragen, die an den Server gehen. Wenn der Start und Ziel die gleichen Ort sind, dann sind es $n^2 - 3n + 2$ Anfragen.
Für jede Anfrage an den Geo-Server für die Entfernungen zwischen zwei Orten A und B sendet der Server eine Antwort mit einem Datensatz zurück. Dieser Datensatz enthält die Entfernung zwischen dem Ort A und dem Ort B in Metern und in Sekunden sowie den Streckenverlauf. Jeder Datensatz und die Null-Werte werden in einem Feld (als Array bezeichnet) gespeichert. Das Array muss dabei $n \times n$ groß sein. Unter dem $n \times n$ Array kann man sich eine Tabelle vorstellen, die n Zeilen und n Spalten besitzt. In der ersten Zeile der Tabelle stehen alle Entfernungen vom Start zu sich selbst, zu den Zwischenzielen und zu dem Ziel. In der letzten Zeile vom Ziel zum Start, zu den Zwischenstopps und zu sich selbst. Und zwischen der ersten und letzten Zeile befinden sich die Zwischenziele. So sind auch die Spalten definiert. Erst kommt der Start, anschließend die Zwischenziele und zum Schluss das Ziel. Die Tabelle wird dann so gelesen, dass man von der Zeile zur Spalte geht und die Entfernung heraus lesen kann. Zum Beispiel möchte man die Entfernung zwischen Start und Ziel erfahren, dann nimmt man die erste Zeile und geht dann in die letzte Spalte. Je nachdem, welche Werte, ob Meter oder Sekunden, für die Berechnung benötigt werden, werden nur diese Werte in der Tabelle bzw. im Array angezeigt. In der Informatik nennt man so etwas Adjazenzmatrix.

Für das Unternehmen im Beispiel erhält man folgende Adjazenzmatrizen für die Entfernungen in Metern und Sekunden:

$c = s$ in m	**H**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**H**
H	0	146	26188	26287	12958	14522	4224	5480	20768	7083	0
F1	∞	0	26100	26199	12870	14434	4072	5328	20680	6995	146
F2	∞	26855	0	2285	25067	12806	31143	32707	19514	20120	26943
F3	∞	26137	2322	0	24349	12088	30425	31989	19213	19402	26225
F4	∞	12885	24293	24392	0	12627	16453	13088	11453	10810	12973
F5	∞	14439	12042	12141	12650	0	18726	20290	10159	7704	14527
F6	∞	3872	31482	31581	16195	19816	0	3423	24006	12377	4024
F7	∞	5325	32253	32352	12974	20587	3816	0	24777	13148	5477
F8	∞	20691	19079	18812	11525	10159	24259	21806	0	14568	20779
F9	∞	6995	19409	19508	10815	7743	11282	12846	14605	0	7083
H	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0

Tabelle 3.2: Adjazenzmatrix mit Entfernungen in Metern zwischen den Orten. [10]

$c = t$ in s	**H**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**H**
H	0	29	1699	1620	825	966	444	590	1180	588	0
F1	∞	0	1654	1575	780	921	340	486	1135	543	29
F2	∞	1735	0	365	1620	944	1977	2165	1272	1324	1797
F3	∞	1572	405	0	1457	781	1814	2002	1158	1161	1634
F4	∞	749	1515	1436	0	783	915	940	705	749	810
F5	∞	971	846	767	855	0	1213	1401	726	559	1033
F6	∞	434	1968	1889	968	1235	0	420	1323	857	467
F7	∞	475	2097	2018	936	1364	469	0	1452	986	508
F8	∞	1127	1181	1112	739	687	1293	1399	0	872	1188
F9	∞	527	1224	1144	750	491	769	957	846	0	589
H	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0

Tabelle 3.3: Adjazenzmatrix mit Entfernungen in Sekunden zwischen den Orten [10]

3.3.2 Routenplanung

Folgende Anforderungen sind gegeben: Es gibt einen Start- und einen Zielort, beide können auch der gleiche Ort sein, sowie eine natürliche Zahl i an unterschiedlichen Zwischenzielen (für $i = 0, 1, 2, ...$, $i \in \mathbb{N}$ und $i < \infty$). Gesucht wird die schnellste oder kürzeste Gesamtstrecke, wobei alle Orte genau einmal besucht werden, es sei denn, der Start und das Ziel haben die selben Koordinaten. Dann wird dieser Punkt als einziger zweimal besucht.
Wird nach der Strecke mit den geringsten Kosten gesucht und Start- und Zielpunkt sind identisch, dann kennt man das Problem bereits aus der theoretischen Informatik als sogenanntes *Travelling Salesman Problem (TSP).*
Es gibt n Orte, sie enthalten Start, Ziel und die i Zwischenziele ($i = n - 2$). Zwischen je zwei Orten A und B gibt es eine Strecke mit den Kosten $c(A, B)$. Dies kann in einem Graphen $G = (V, E)$ veranschaulicht werden, indem die Orte durch Knoten $v \in V$, die Strecken zwischen den Orten durch Kanten $e \in E$ und die Streckenlängen bzw. Fahrzeiten durch die Kosten $c\colon E \to \mathbb{N}$ repräsentiert werden. [4]

Benötigt wird eine Funktion, die möglichst performant arbeitet und nicht berücksichtigen muss, ob es sich bei der Eingabe um ein TSP handelt oder nicht.
Man benötigt keine Informationen von irgendeinem Ort zum Start und auch nicht vom Ziel zu irgendeinem anderen Ort sowie von einem Ort zu sich selbst. Es werden zwei Felder benötigt, ein Array für die kürzeste Strecke und ein Array für die schnellste Strecke. Alle nicht benötigten Zellen initialisiert man bei der Erzeugung der Arrays mit Null. Die anderen Zellen müssen nach der Abfrage an den Server einen Wert größer Null haben. Ausnahme ist, wenn der Start gleich dem Ziel ist, dann ist als einziges der Weg vom Start zum Ziel Null.
Da die Positionen vom Start (erste Stelle im Array) und vom Ziel (letzte Stelle im Array) fest vorgegeben sind, müssen nur die unterschiedlichen Möglichkeiten zwischen den i Zwischenzielen zum Erhalten der gewünschten Route berücksichtigt werden. Somit gibt es $i!$ Möglichkeiten. Eine Teilroute $(A_1, A_2, ..., A_i)$ ist wieder durch eine Permutation $\pi\colon \{1, 2, ..., i\} \to \{1, 2, ..., i\}$ der i Zwischenstopps mit $\pi(j) = A_j, j = 1, ..., i$, gegeben. Die Länge ε der gesamten Route ergibt sich unter der Berücksichtigung, dass $c(A, B) \neq c(B, A)$ ist (siehe Abschnitt 2.5), wie folgt:

$$\varepsilon = \min_{\pi} \left(\sum_{a=1}^{i-1} c(\pi(a), \pi(a+1)) \right) + c(Start, \pi(1)) + c(\pi(i), Ziel))$$

Die Länge ε der kürzesten Route wird als Lösung ausgegeben. [4]

Programmierung

In dem PHP-Code gibt es eine Hauptfunktion (*calculateRoute*) und zwei Hilfsfunktionen (*deleteInArray* und *factorial*).

An die Hauptfunktion muss ein $n \times n$ Array (Adjazenzmatrix) übergeben werden, welches vor der Übergabe an die Hauptfunktion erzeugt wird. Das Array enthält n Arrays, für jeden Ort eins, die auch jeweils die Länge n haben und die einzelnen Streckenlängen oder -zeiten zu den n Orten enthalten. Die Größe des übergebenen Arrays wird geprüft. Ist diese kleiner als Zwei, gibt es also entweder keinen oder nur einen Ort, so ist die benötigte Länge bzw. Zeit (Kosten) für die Route Null. Wenn das Feld größer als Eins und kleiner als Vier ist, dann ist ein Start und ein Ziel vorhanden, mit oder ohne Zwischenziel. Für beide Möglichkeiten gibt es jeweils nur eine Lösung ($\varepsilon = c(Start, Ziel)$ oder $\varepsilon = c(Start, \pi(1)) + c(\pi(1), Ziel)$). Gibt es mehr als drei Orte (Start, Ziel und i Zwischenziele $i \in \{2, 3, ...\}$, dann wird jede der $i!$ möglichen Routen getestet. Dazu wird an die Funktion *factorial* das $n \times n$ Array mit den enthaltenen Daten, ein Array mit der Länge n (Anzahl aller Orte) mit Durchnummerierung der Orte von 0 bis $(n-1)$, die Position des letzten Ortes, die Entfernung bis zum letzten Punkt, ein Array mit der Reihenfolge der zuvor besuchten Orte, ein Zahlenwert für die kürzeste Route und noch ein Array mit dem Verlauf der kürzesten Route übergeben.

In der Funktion *factorial* wird als erstes mittels der Länge des Feldes, welches zu Beginn von 0 bis $(n-1)$ durchnummeriert war, geprüft, ob alle Zwischenziele besucht worden sind. Ist das Array größer als zwei, dann wurde mindestens ein Zwischenhalt nicht besucht und alle Streckenkombinationen mit dem fehlenden Punkt bzw. den fehlenden Punkten sind noch zu überprüfen. Nun schaut man, ob die Teilstrecke bis zu diesem Punkt weniger Kosten verursacht. Wenn ja, ruft man die Funktion *factorial* erneut auf. Dabei wird wieder das Feld mit allen Daten zwischen den Orten, das Array der Orte ohne die bereits besuchten Orte und dem letzten Ort, die Teilstrecke bis zum letzten Punkt, die Route der Teilstrecke, die Länge der Route mit den niedrigsten Kosten und dem Verlauf dessen Route übergeben. Wenn noch keine effiziente Route ermittelt werden konnte, dann wird auch die Funktion *factorial* aufgerufen. Ist die Feldlänge gleich zwei, dann bleiben nur noch der Start und das Ziel übrig. Der Start wird

am Anfang berücksichtigt. Deswegen muss nur noch die Distanz zwischen dem letzten Zwischenstopp und dem Ziel zu der Teilstrecke addiert werden, um die Gesamtlänge der Route zu erhalten.
Sind alle Kombinationen durchprobiert, so gibt die Funktion *factorial* den kürzesten bzw. schnellsten Weg mit Verlauf an die Funktion *calculateRoute* zurück. Diese liefert diese Werte mit den Schlüsseln „distance“ und „route“ an den Nutzer.
Die Hilfsfunktion *deleteInArray* wird von *factorial* aufgerufen, bevor *factorial* sich selbst erneut aufruft. Dabei wird das Array der Größe k mit einem Schlüssel (auch als *Key* bezeichnet) an der Stelle, wo ein Element in dem Array gelöscht werden soll, an die Funktion übermittelt. Als Rückgabe erhält man das Array mit der Länge $(k-1)$ und ohne dem Element an der vorigen Stelle des Schlüsselwertes.

```
1  function deleteInArray($inputArray, $key)
2  {
3      if($key > count($inputArray))
4      {
5          return $inputArray;
6      }
7      else if(is_array($inputArray))
8      {
9          $returnArray = [];
10
11         for($i = 0; $i < count($inputArray); $i++)
12         {
13             if($i != $key)
14             {
15                 array_push($returnArray, $inputArray[$i]);
16             }
17         }
18
19         return $returnArray;
20     }
21 }
22
23 function factorial($data, $poiArray, $position = 0, $distance = 0, $route = [],
       &$shortesDistance = 0, &$shortesRoute = [])
24 {
25     if(count($poiArray) > 2)
26     {
27         for($i = 1; $i < (count($poiArray) - 1); $i++)
28         {
29             if($shortesRoute > ($distance + $data[$position][$poiArray[$i]])
                   || $shortesRoute == 0)
30             {
31                 $copyData = deleteInArray($poiArray, $i);
32                 $copyRoute = $route;
33                 array_push($copyRoute, $position);
34
35                 factorial($data, $copyData, $poiArray[$i], ($distance +
                       $data[$position][$poiArray[$i]]), $copyRoute,
                       $shortesDistance, $shortesRoute);
36             }
37         }
```

```
38      }
39      else if(count($poiArray) == 2)
40      {
41          if($shortesDistance > ($distance + $data[$position][(count($data) -
                1)]) || $shortesDistance == 0)
42          {
43              $shortesRoute = $route;
44              array_push($shortesRoute, $position,(count($data) - 1));
45              $shortesDistance = $distance + $data[$position][(count($data) -
                    1)];
46          }
47      }
48
49      return array($shortesDistance, $shortesRoute);
50  }
51
52  function calculateRoute($inputArray)
53  {
54      $distance = 0;
55      $route = [];
56
57      if(count($inputArray) < 2)
58      {
59          $distance = 0;
60          if(count($inputArray) == 0)
61          {
62              $route = [];
63          }
64          else
65          {
66              $route = [0];
67          }
68      }
69      else if(count($inputArray) > 1 && count($inputArray) < 4)
70      {
71          for($i = 0; $i < (count($inputArray) - 1); $i++)
72          {
73              array_push($route, $i);
74          }
75
76          array_push($route, (count($inputArray) - 1));
77      }
78      else
79      {
80          array_push($route, 0);
81          $poiArray = [];
82
83          for($i = 0; $i < (count($inputArray)); $i++)
84          {
85              array_push($poiArray, $i);
86          }
87
88          $answer = factorial($inputArray, $poiArray);
89
90          $distance = $answer[0];
91          $route = $answer[1];
92
93      }
94
95      return array(
96          'distance' => $distance,
97          'route' => $route
```

```
98      );
99  }
```

Berechnung am Beispiel

Wenn es i Zwischenziele gibt, dann gibt es $i!$ mögliche Kombinationen, so dass man alle Orte einmal besucht.
Sobald alles berechnet und die kürzeste oder schnellste Route ermittelt wurde, wird der Verlauf in die Karte gezeichnet.

$$\begin{aligned}
\varepsilon_s &= c(H,F1) + c(F1,F6) + c(F6,F7) + c(F7,F4) + c(F4,F8) \\
&\quad + c(F8,F2) + c(F2,F3) + c(F3,F5) + c(F5,F9) + c(F9,H) \\
&= 146 + 4072 + 3423 + 12974 + 11453 + 19079 + 2285 + 12088 + 7704 \\
&\quad + 7083 \\
&= 80307
\end{aligned}$$

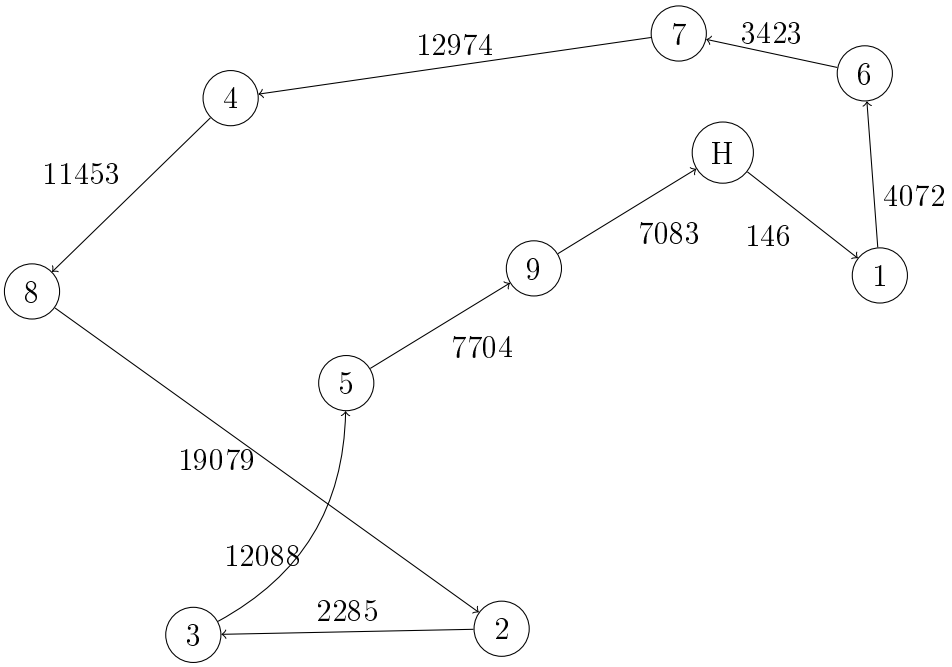

Abbildung 3.6: Errechneter Routenverlauf (kürzeste Entfernung) in Graphendarstellung

$$\begin{aligned}
\varepsilon_t &= c(H,F1) + c(F1,F6) + c(F6,F7) + c(F7,F4) + c(F4,F8) \\
&\quad + c(F8,F2) + c(F2,F3) + c(F3,F5) + c(F5,F9) + c(F9,H) \\
&= 29 + 340 + 420 + 936 + 705 + 1181 + 365 + 781 + 559 + 589 \\
&= 5905
\end{aligned}$$

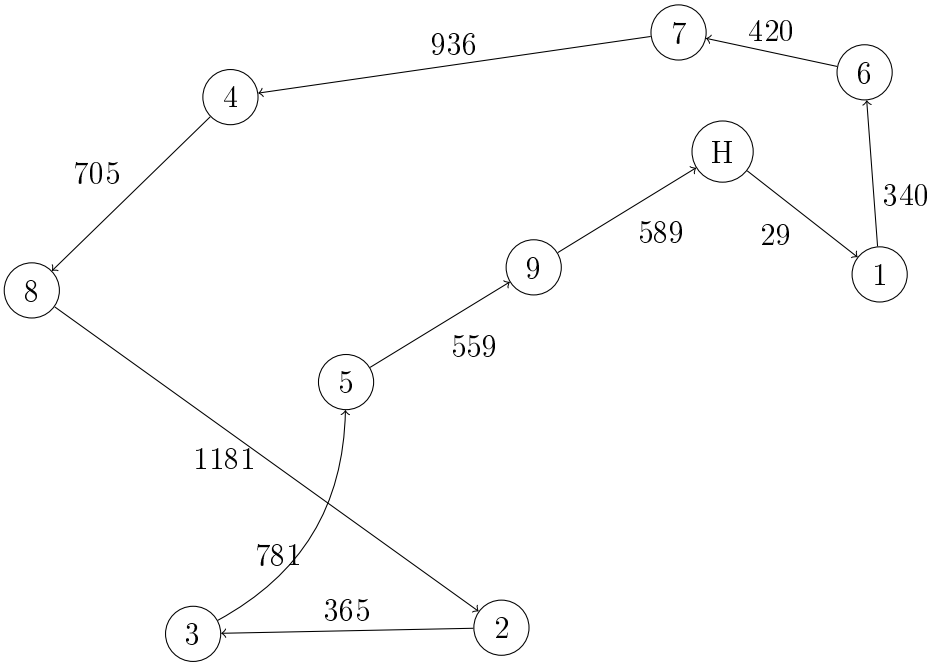

Abbildung 3.7: Errechneter Routenverlauf (kürzeste Zeit) in Graphendarstellung

Nachdem beide Arrays das Programm durchlaufen haben, sind hier jeweils die gleichen Routen herausgekommen.

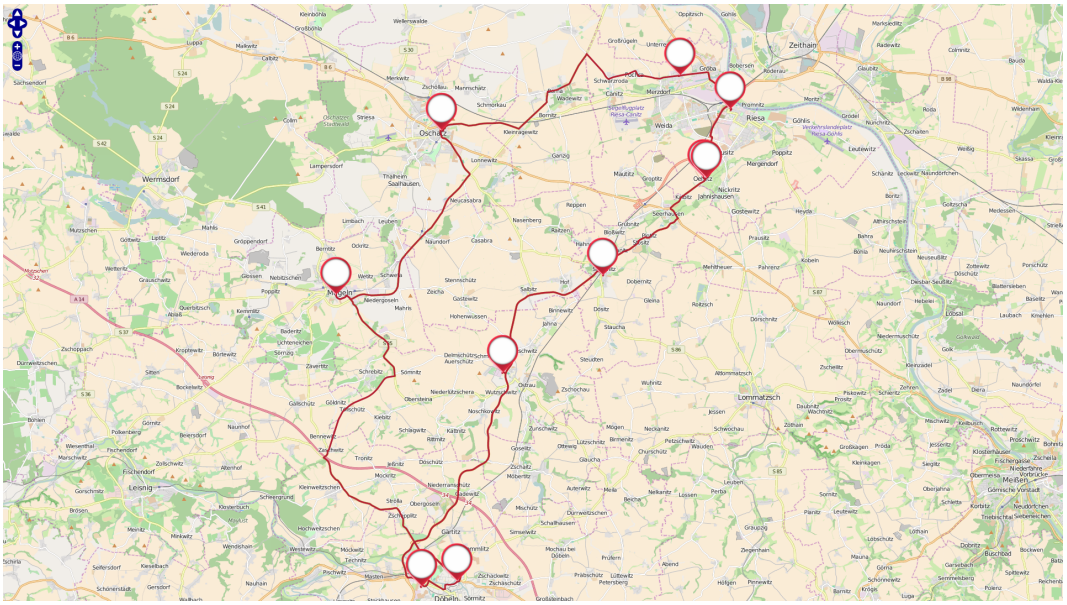

Abbildung 3.8: Errechneter Routenverlauf in Karte.

Der Algorithmus berechnete die Strecke mit 80307 Metern (≈ 80,3 Kilometer) und die Zeit mit 5905 Sekunden (≈ 98 Minuten; ≈ 1,6 Stunden). Weil die zeitliche und streckentechnische Tour dieselbe sind, kann auch die Durchschnittsgeschwindigkeit berechnet werden.

$$\begin{aligned} v_{\varnothing} &= \frac{\varepsilon_s}{\varepsilon_t} \\ &= \frac{80307}{5905}\frac{m}{s} \\ &\approx 13.6\frac{m}{s} \end{aligned}$$

Die errechnete Durchschnittsgeschwindigkeit für die Tour beträgt zirka 13,6 $\frac{m}{s}$ (≈ 49 $\frac{km}{h}$). Ob diese ermittelte Geschwindigkeit stimmt, wird im späteren Experiment bei der Auswertung (3.3.4) überprüft.

3.3.3 Datenerhebung

Webanwendung und Applikation (App)

Für die Erhebung der Daten von Personen oder Objekten wurde eine Webanwendung und eine Applikation (kurz „App“) entwickelt. Diese Anwendungen greifen mittels einem Script auf den GPS-Sensor zu, falls dieser vorhandenen ist. Diese Daten enthalten die lon- und lat-Koordinaten. Zu diesen Daten werden eine Erkennungs-ID, der Zeitstempel und gegebenenfalls zusätzliche Informationen hinzugefügt. Die erhobenen Daten werden anschließend zur

Speicherung an einen Server übergeben.

Geo-Server

Für die Speicherung der Daten (Geo-Daten) wurde ein Server eingerichtet. Dieser Server enthält eine Datenbank, in der die Geo-Daten und zusätzliche Informationen (ID, lon, lat, Zeitstempel) gespeichert werden. Zudem wird zu dem Server ein Zugang eingerichtet, so dass von außen auf diese zugegriffen werden kann und die Daten übergeben werden können. Die Webanwendung oder Applikation kommuniziert in regelmäßigen Zeitabständen mit dem Server und übergibt alle bis dahin dokumentierten Daten. Ist keine Übertragung an den Server möglich, so werden die erhobenen Daten in den Anwendungen gesichert, bis wieder eine Übertragung möglich ist.

3.3.4 Routenverfolgung und -auswertung

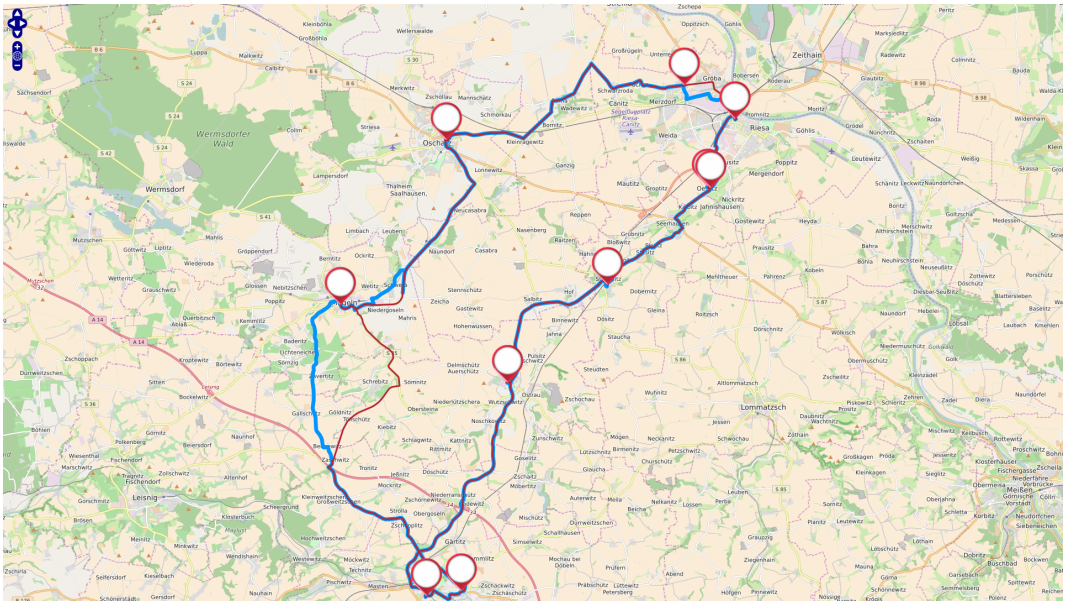

Abbildung 3.9: Karte mit errechneter Strecke und gefahrener Strecke

In der Grafik wird die geplante Route (rot) und die tatsächlich gefahrene Route (blau) dargestellt. Auf der abgefahrenen Strecke befanden sich zwei Baustellen in Riesa und auf der B169 bei Döbeln und ein Verkehrsunfall vor Mügeln. Zwischen Mügeln und Döbeln wurde aufgrund von Vorkenntnissen eine andere Strecke genommen. In Oschatz muss bei der Filiale bis zur Hauptstraße anders gefahren werden, da an der geplanten Stelle ein Verkehrszeichen mit vorgeschriebener Fahrtrichtung nach rechts steht (siehe nächste Grafik). Die Rundreise mit zirka 85 Kilometern (Messinstrument: Tachometer) wurde in einer Stunde, 38 Minuten und 10 Sekunden (Serverdaten) absolviert. Im

Vergleich zwischen geplanter und gefahrener Strecke musste ich zirka fünf Kilometer (+6,25%) mehr fahren als angedacht, war aber dafür 15 Sekunden (-0,25%; ohne 10 Sekundenpause pro Halt: -1,78%) schneller, obwohl an jedem Zwischenstopp mindestens 10 Sekunden gehalten wurde, damit der Server den Stopp eindeutig registrieren konnte. Die Durchschnittsgeschwindigkeit betrug damit zirka 14, 4 Meter in der Sekunde ($\approx 52\frac{km}{h}$) (+6,11%).
Insgesamt wurden 1193 Standorte auf der Rundreise aufgezeichnet und davon werden 1089 Orte für die Darstellung in der Webanwendung genutzt.

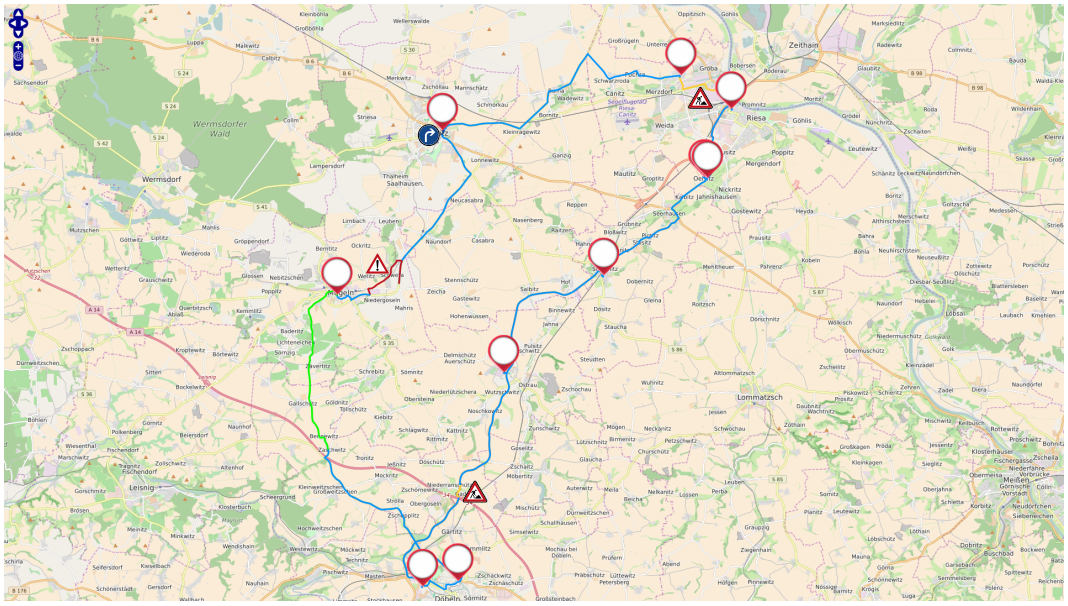

Abbildung 3.10: Gefahrene Rundreise mit Informationen zu den Abweichungen [11] [12] [13]

Die Schlussfolgerung aus diesem Experiment ist, dass die zuvor ermittelten theoretischen Werte für die Strecke, die Zeit und Durchschnittsgeschwindigkeit mit etwas Abweichung nach oben mit den in der Praxis ermittelten Werten übereinstimmen. Wahrscheinlich hätte die Zeit bei normalem Verkehr und ohne Umleitungen genau gereicht, aber bei hohem Verkehrsaufkommen eher nicht.

3.3.5 Geofencing

Bedeutung

Der Begriff „Geofencing“ ist ein Kunstwort und besteht aus einer Kombination von „geography“ (engl. Geographie) und „fencing“ (engl. Einzäunung). Mit Geofencing besteht die Möglichkeit Personen und Objekte mithilfe eines imaginären Zauns in lokalen Bereichen zu halten. Befinden sich überwachte Personen und Objekte nicht mehr im erlaubten geographischen Objekt, schlägt

das Geofencing-System Alarm und informiert betroffene Benutzer. [14]

Algorithmus

Wenn auf einer Karte ein geographischer Bereich abgegrenzt ist und nun die Frage im Raum steht, ob sich eine Person oder ein Objekt (gekennzeichnet als Punkt) in diesem markierten Bereich befindet oder nicht, dann kann durch Visualisierung und Interpretation diese beantwortet werden. Auch ein Computer kann diese Frage lösen, jedoch benötigt er dazu Algorithmen. Die Maschine kann im Gegensatz zum Menschen auf Anhieb die geometrische Form des eingezäunten Objekts nicht erkennen, außer diese wird auch an diese übermittelt. Man geht davon aus, dass die Form unbekannt ist.
Man geht davon aus, dass jedes Geofencing-Objekt (Rechteck, Kreis, Oval, n-Eck, usw.) als Polygon gespeichert und dargestellt werden kann. Die Anzahl der erforderlichen Punkte zum Zeichnen des Polygons ist dabei größer als zwei und kleiner als unendlich.

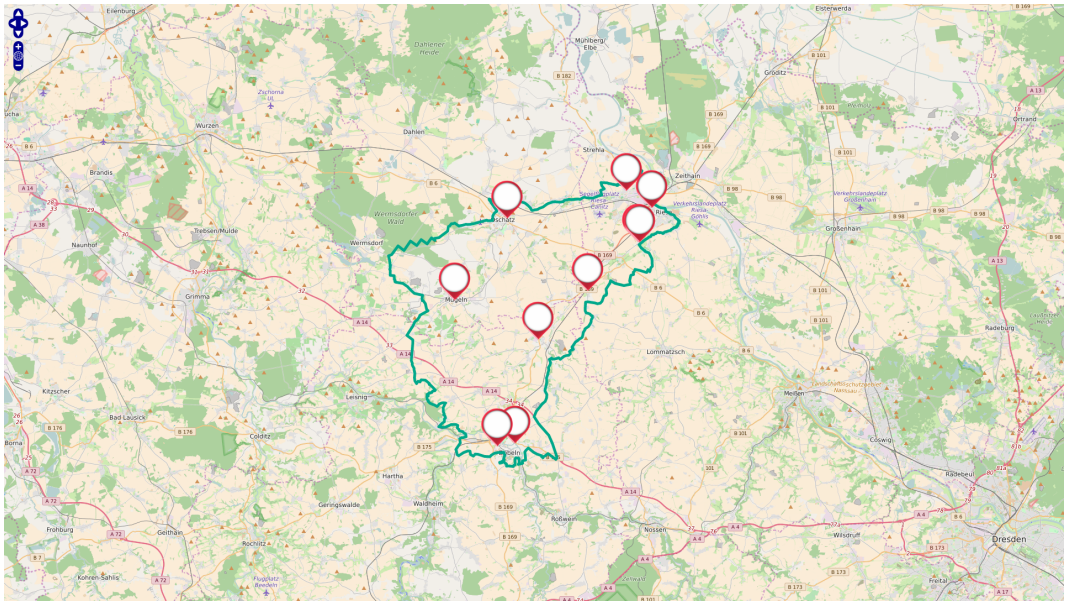

Abbildung 3.11: Eingezäunter Bereich (Polygon)

Bounding Box

Es wird sozusagen ein Rechteck um das Polygon gelegt, welches die minimalen und maximalen Werte des Polygons auf der x- und y-Achse bestimmt. Jeder Eckpunkt des Polygons muss überprüft werden.
Jeder Punkt $P_0, P_1, ..., P_{n-1}$ des Polygons muss dabei betrachtet werden. Ist noch kein minimaler und maximaler x-Wert und kein minimaler und maximaler y-Wert ermittelt, dann fand noch keine Überprüfung von irgendeinem Punkt statt. Trifft dieser Fall ein, dann beginnt man mit dem Punkt P_0 und

setzt $x_{min} = x_{max} = P_{0_{lon}}$ und $y_{min} = y_{max} = P_{0_{lat}}$. Alle anderen Punkte P_i (für $i = 1, 2, ..., n-1$) werden anschließend immer mit den x- und y-Werten verglichen. Wenn $x_{min} > P_{i_{lon}}$ ist, dann setzt man $x_{min} = P_{i_{lon}}$, sonst bleibt der Wert von x_{min} bestehen. Ebenso, falls $x_{max} < P_{i_{lon}}$ oder $y_{min} > P_{i_{lat}}$ oder $y_{max} < P_{i_{lat}}$ ist, dann setzt man $x_{max} = P_{i_{lon}}$ oder $y_{min} = P_{i_{lat}}$ oder $y_{max} = P_{i_{lat}}$, sonst bleiben die Werte wie sie sind. Nachdem alle Punkte geprüft und von den x- und y-Werten jeweils das Minimum und das Maximum ermittelt wurden, vergleicht man, ob der zu prüfende Punkt $P = \binom{P_{lon}}{P_{lat}}$ in dem Rechteck liegt oder nicht. Falls $(x_{min} \leq P_{lon} \leq x_{max}) \wedge (y_{min} \leq P_{lat} \leq y_{max})$ gilt, dann liegt P im Rechteck sonst nicht.
Liegt der Punkt im Rechteck, dann muss genauer getestet werden, ob er sich wirklich im Polygon befindet, und wenn nicht, dann befindet er sich mit Sicherheit nicht im Polygon.

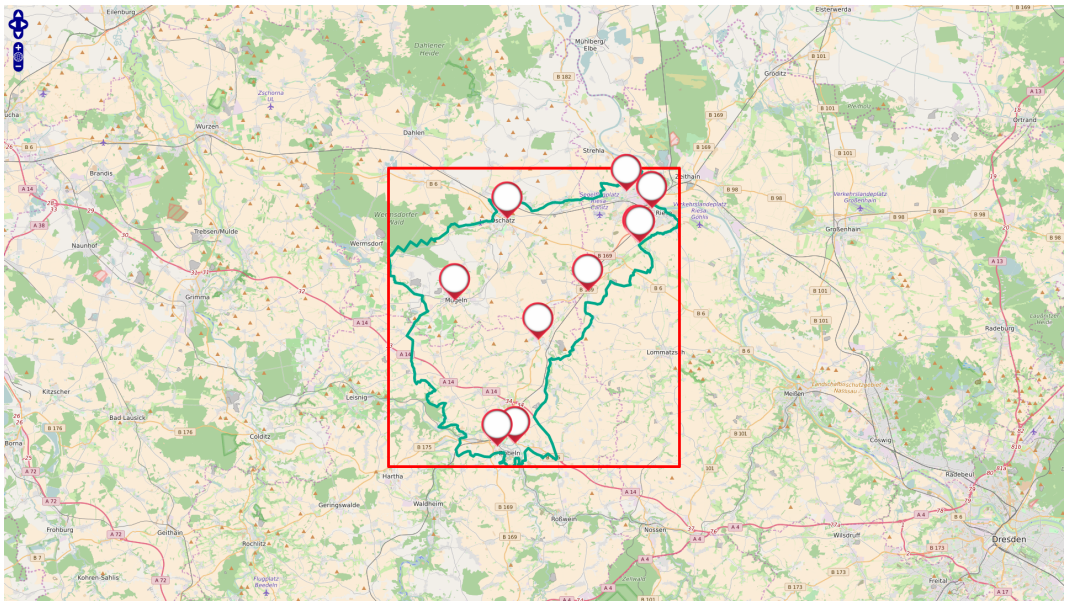

Abbildung 3.12: Bounding Box vom Polygon

Punkt-in-Polygon-Test nach Jordan

Mit diesem Algorithmus kann man relativ schnell bestimmen, ob sich ein Punkt in einem Polygon befindet. Vom Punkt geht ein Strahl in eine beliebige Richtung. Nun wird die Anzahl der Schnittpunkte δ zwischen dem Strahl und dem Polygonrand ermittelt. [15]

Berechnung

Von dem Punkt P aus muss ein Strahl erzeugt werden.

$$\overrightarrow{S_P} = \begin{pmatrix} P_{lon} \\ P_{lat} \end{pmatrix} + x \cdot \begin{pmatrix} z_x \\ z_y \end{pmatrix}$$

z_x und z_y sind Zufallszahlen, wobei entweder $(-1 < z_x, z_y < 0)$ oder $(0 < z_x, z_y < 1)$ gilt und für x gilt: $(0 \leq x < \infty)$. Der Wert für x wird mit 1 festgelegt, um einen weiteren Punkt zu erzeugen durch den der Strahl geht.

Schnittpunkte zwischen Strahl und Rand

Anschließend wird der erzeugte Strahl auf einen Schnittpunkt bei jedem Teilrand untersucht. Die Schnittpunkte werden wie bei der Schleife im Kapitel „Routenverfolgung“ bei „Berechnungen“ berechnet. Jeder Schnittpunkt wird gezählt. Wenn die Anzahl der Schnittpunkte ungerade ist, dann ist der Punkt P im Polygon. Wenn die Anzahl gerade oder gleich Null ist, dann liegt der Punkt außerhalb des Polygons. Da das Polygon eine geschlossene Fläche ist und es verlassen möchte, so muss man mindestens einmal den Rand überqueren. Wird anschließend wieder über die Grenze gegangen, so befindet man sich wieder im Polygone. Es besteht aber auch die Möglichkeit, dass unendlich viele Schnittstellen gefunden werden, so kann der Strahl auf einem Teilrand liegen und der Strahlentest muss wiederholt werden, in dem die Richtung des Strahls geändert wird.

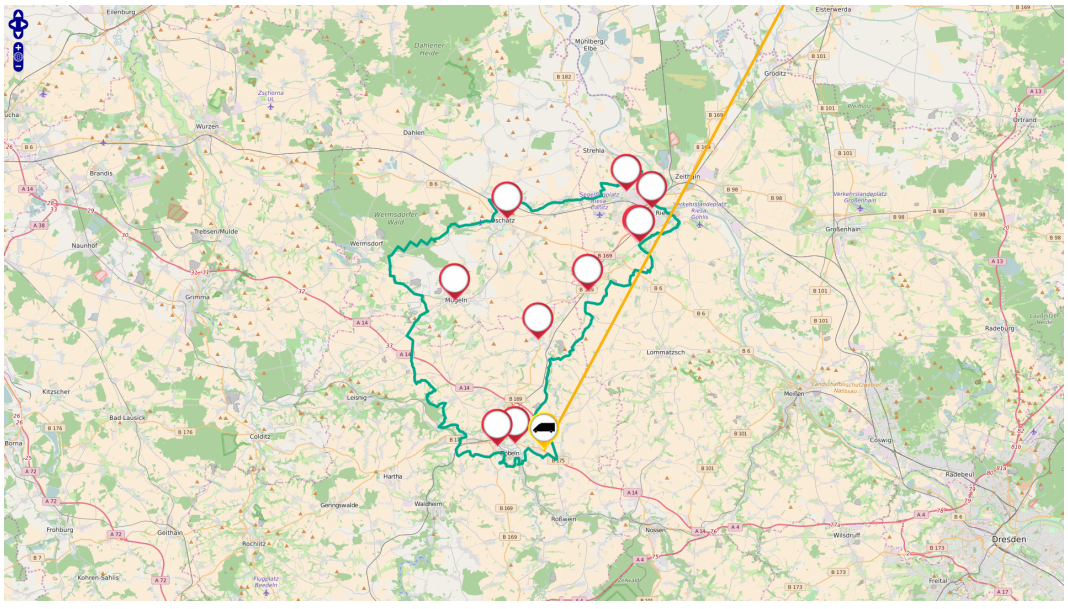

Abbildung 3.13: Test nach Jordan für Polygon

Nachteil vom Strahlentest

Nachteil von diesem Test sind die vielen Berechnungen, Vergleiche und Überprüfungen. Hinzu kommen noch Fallunterscheidungen, wenn der Strahl einen Teilrand schneidet.

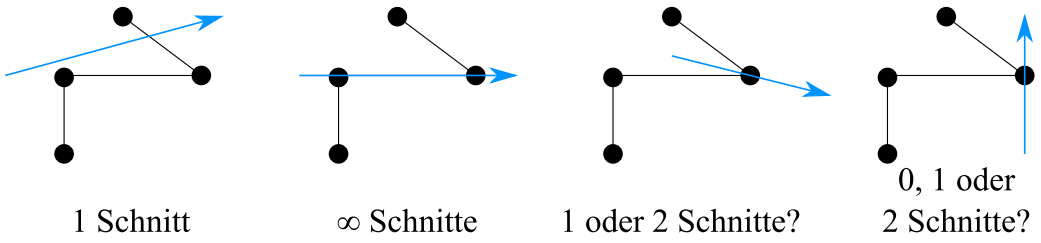

Abbildung 3.14: Möglichkeiten Schnittstellen [16]

Im ersten Bild in Abbildung 3.14 ist der Fall klar, denn es gibt nur einen Schnittpunkt. Unendlich viele Schnittstellen sind in der zweiten Darstellung zu finden. Es kann keine eindeutige Aussage darüber gegeben werden, ob dann der Punkt sich im Polygon befindet. Der ganze Strahlentest muss wiederholt werden. Das Wiederholen verringert die Effizienz des Algorithmus und verursacht lange Berechnungszeiten. In der dritten Illustration im Bild ist es für den menschlichen Verstand klar, dass der Strahl den Rand des Polygons genau einmal in einem Eckpunkt schneidet. Mathematisch gesehen berühren sich der Strahl und zwei Teilränder. Somit existieren zwei Schnittpunkte. Im vierten Bild berührt der Strahl einen Eckpunkt vom Polygon. Wieder sind es aus mathematischer Sicht zwei Schnittpunkte. Technisch gesehen ist es eine Berührung, aber kein Schnittpunkt, da der Strahl nicht den Rand durchquert. Es muss in den Bildern 3 und 4 immer eine Fallunterscheidung stattfinden, ob der Strahl den Rand durchquert und wenn dieses eintrifft, wie oft wirklich.
Wie bereits erwähnt, verursacht dies hohe zeitliche Kosten bei der Berechnung. Also muss eine noch einfachere Methode gefunden werden, die bestimmt, ob ein Punkt im Polygon liegt oder nicht.

Ansatz mit Hilfe von Computergraphik

Eine Idee zum Lösen des „Punkt im Polygon-Problems“ ist die Verwendung der baryzentrischen Koordinaten im Dreieck.

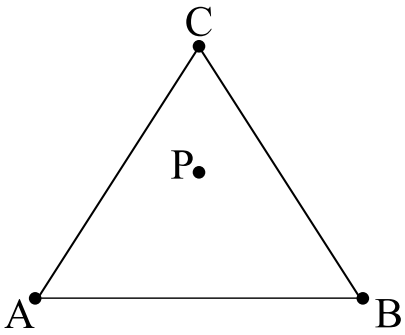

Abbildung 3.15: Dreieck mit Punkt P [17]

Die Punkte A, B und C bilden ein Dreieck. Zusätzlich gibt es den Punkt P, welcher sichtbar im Dreieck liegt. Bekannt von A, B und C sind nur die x- und y-Koordinaten, also geographische Länge und Breite. Weitere Informationen sind nicht bekannt. Somit kann nur der Umkreis oder der Flächeninhalt des Dreiecks berechnet werden. Da der Umfang auch keine weitere Hilfe ist, muss der Flächeninhalt A_{ABC} zum Lösen des Problems „Punkt im Polygon“ behilflich sein.

$$\begin{aligned} A_{ABC} &= |\frac{1}{2} \cdot det(\overrightarrow{AB}, \overrightarrow{BC})| \\ &= |\frac{1}{2} \cdot ((B_{lon} - A_{lon}) \cdot (C_{lat} - A_{lat}) - (B_{lat} - A_{lat}) \cdot (C_{lon} - A_{lon}))| \end{aligned}$$

Da bekannt ist, dass der Punkt P im Dreieck liegt und die Größe des Flächeninhalts ebenso bekannt ist, kann folgende Formel aufgestellt werden:

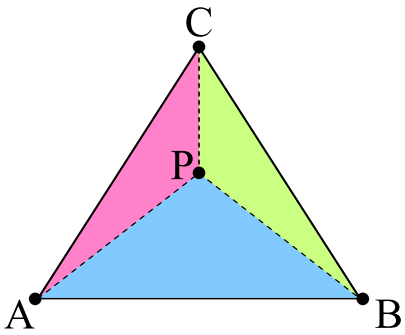

Abbildung 3.16: Dreieck unterteilt in Teildreiecke mit Punkt im Dreieck [17]

$$\begin{aligned} A_{ABC} = A_{ABCP} &= A_{ABP} + A_{BCP} + A_{CAP} \\ P &= \alpha A + \beta B + \gamma C \\ \alpha + \beta + \gamma &= 1 \\ 0 &\leq \alpha, \beta, \gamma \leq 1 \end{aligned}$$

Wenn das Dreieck ABC in drei Teildreiecke ABP, BCP und CAP unterteilt wird und der Punkt P liegt im Dreieck ABC, dann ist der Flächeninhalt des Dreiecks ABC genau so groß wie die Summe der drei Teildreiecke ABP, BCP und CAP. Bedingung ist, dass α, β und γ größer oder gleich Null sind. Daraus folgt, wenn der Punkt P nicht im Dreieck ist, so ist der Flächeninhalt vom Dreieck ABC auch gleich der Summe der Flächeninhalte der drei Teildreiecke ABP, BCP und CAP, aber mindestens eins von α, β und γ ist kleiner 0.

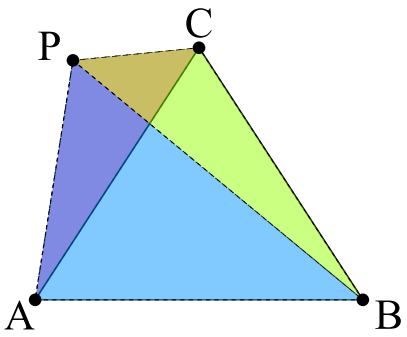

Abbildung 3.17: Dreieck unterteilt in Teildreiecke mit Punkt außerhalb vom Dreieck [17]

Nachteil der baryzentrischen Koordinaten

Dieses Verfahren ist rechnerisch einfach und lässt sich auch auf Polygone anwenden, jedoch nur auf konvexe Polygone. Auch konkave Polygon müssen berücksichtigt werden und deswegen ist dieser Algorithmus allgemein nicht zu gebrauchen.

Winkelsummen-Algorithmus

Nachdem auch die baryzentrischen Koordinaten nicht beim Lösen des vorhanden Problems helfen, muss es mit dem folgenden Versuch endlich funktionieren. Gegeben ist wieder ein Dreieck ABC in dem ein Punkt P ist. P ist der Mittelpunkt von dem Kreis der um das Polygon gezogen wird, so dass alle Randpunkte vom Polygon im Kreis liegen. Wenn die Summe aller Winkel ϕ zwischen je zwei aufeinander folgenden Randpunkten R und dem Punkt P 360° beträgt, dann wird damit bewiesen, dass der Punkt P im Polygon ist.

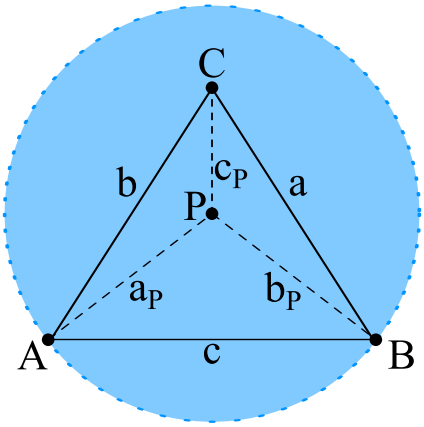

Abbildung 3.18: Dreieck mit Umkreis

$$
\begin{aligned}
a_P &= A - P \\
b_P &= B - P \\
c_P &= C - P \\
R &= (a_P, b_P, c_P) \\
\phi_{Gesamt} &= \sum_{i=1}^{n-1} \phi_i = \sum_{i=1}^{n-1} \arccos\left(\frac{(R(i)-P)\cdot(R(i+1)-P)}{|(R(i)-P)|\cdot|(R(i+1)-P)|}\right)
\end{aligned}
$$

Ist der absolute Betrag von ϕ_{Gesamt} gleich 360°, dann liegt der Punkt im Polygon, andernfalls nicht.
Diese Formel funktioniert für das Beispiel am Dreieck, aber nicht für alle Polygone allgemein. Eine Fallunterscheidung muss getroffen werden, denn der Richtungsverlauf muss berücksichtigt werden. Dafür wird die Richtungsbestimmung von den baryzentrischen Koordinaten verwendet $(\det(\overrightarrow{PR(i)}, \overrightarrow{PR(i+1)}))$. Ist das Ergebnis im positiven Zahlenbereich, so ist die Richtung gegen den Uhrzeigersinn und der Winkel ϕ ist auch positiv, andernfalls verläuft die Richtung im Uhrzeigersinn und der Winkel ϕ ist negativ.

$$
\phi_{Gesamt} = \sum_{i=1}^{n-1} \begin{cases} -\arccos(\frac{(R(i)-P)\cdot(R(i+1)-P)}{|(R(i)-P)|\cdot|(R(i+1)-P)|}) & \text{für } 0 > \det(\overrightarrow{PR(i)}, \overrightarrow{PR(i+1)}) \\ \arccos(\frac{(R(i)-P)\cdot(R(i+1)-P)}{|(R(i)-P)|\cdot|(R(i+1)-P)|}) & \text{für } 0 \leq \det(\overrightarrow{PR(i)}, \overrightarrow{PR(i+1)}) \end{cases}
$$

Der absolute Betrag von Winkel $|\phi_{Gesamt}|$ wird dann verglichen. Ist $|\phi_{Gesamt}| = 360°$, dann befindet sich der Punkt P im Polygone. Wenn $|\phi_{Gesamt}| < 360°$ ist, dann befindet sich der Punkt P außerhalb vom Polygone.

3.3.6 Auswertung Fallstudie

Die Fallstudie hat gezeigt, dass die Ergebnisse zwischen Theorie und Praxis nur wenig voneinander abweichen. Wie die Praxis zeigt, sind Umleitungen durch Baustellen oder Unfälle zu berücksichtigen, aber auch falsche Informationen auf den Geo-Server verursachen bei der Planung Fehler. So ein Fehler passierte zum Beispiel bei der Filiale 4 (F4). Dort soll nach der Filiale 4 in Richtung Filiale 8 nach links abgebogen werden. Das Verkehrszeichen an der Kreuzung zeigt an, dass nur nach rechts abgebogen werden darf.
Das Aufzeichnen der absolvierten Rundreise verlief ohne Probleme und der Server konnte sämtliche Daten in der Datenbank speichern. Diese könnten anschließen durch Einsetzen der Algorithmen und Berechnungen, wie im Kapitel 2 beschrieben, in einem Verlauf veranschaulicht werden.
Am Ende dieser Fallstudie ist festzustellen, dass die Fallstudie erfolgreich umgesetzt werden konnte.

4. Zusammenfassung

Das Ziel eine webbasierte Anwendung zur Routenplanung und -optimierung zu entwickeln wurde erreicht. Mit Algorithmen können effiziente Routen berechnet werden, so dass Unternehmen Treibstoffkosten und Arbeitsentgelte einsparen können und somit den Profit des Unternehmens steigern. Eine Kontrolle über die Einhaltung der Routen bietet die Anzeige der Routenverläufe und für das Einhalten der erlaubten Gebiete das Geofencing. Am Beispiel eines Betriebes zeigte sich, dass die theoretisch vorgegebenen optimalen Fahrzeiten und Fahrtstrecken annähernd gleich den in der Praxis ermittelten Werten sind.

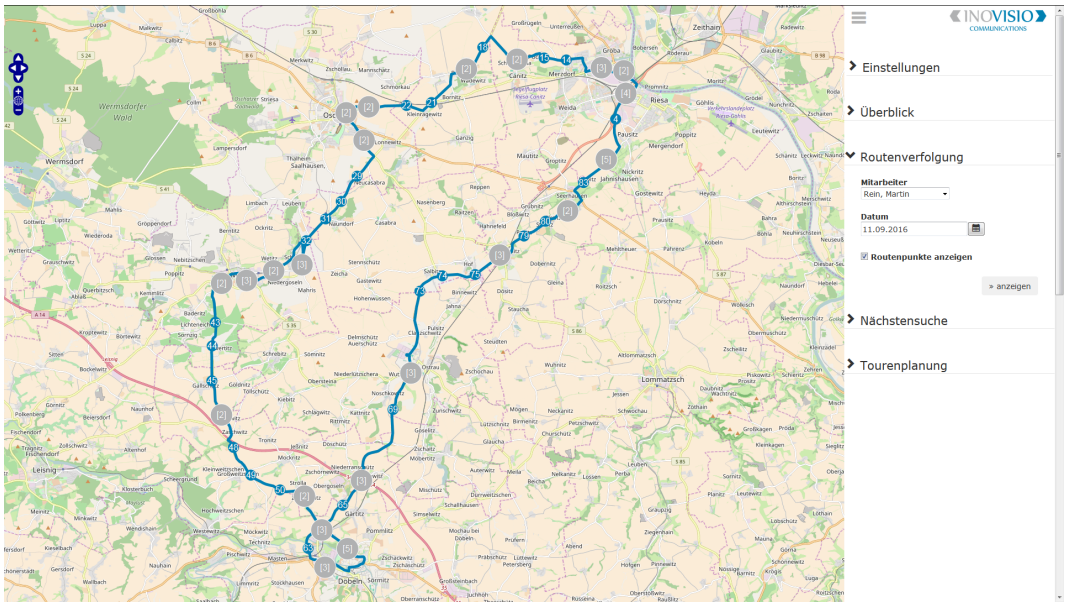

Abbildung 4.1: Webanwendung mit Fallbeispiel

Es steht nach dieser Arbeit fest, egal wie gut man es versucht alle Fälle abzudecken, es wird einem nicht gelingen, aber der Versuch das Beste herauszuholen benötigt Erfahrungen und Rückschläge. Auch musste ich persönlich die Erfahrung sammeln, dass zu kompliziertes Denken die einfachen Lösungen einen übersehen lassen. Weiterhin müssen Verbesserungen und Änderungen vorgenommen und weiterentwickelt werden. So könnte man mit den gesammelten Daten Bewegungsprofile erstellen und auswerten und diese für Zielvorschläge nutzen.
Im Hinblick auf die Ethik sind neben dem Nutzen für die Unternehmen natürlich auch immer der Eingriff in die Privatsphäre der Mitarbeiter und die Einhaltung datenschutzrelevanter Aspekte zu beachten. Es ist wichtig, dass der Zugang zu allen gespeicherten Punkte innerbetrieblich vertrauensvoll behandelt werden (Datensicherheit) und nur diejenigen Einsicht nehmen können, die es

betrifft (Datenschutz). Auch sollten die Nutzer zu einer solchen Anwendung Vertrauen haben und die Entwickler das Vertrauen nicht ausnutzen und weiterhin heimlich Daten aufzeichnen, obwohl die Anwendung deaktiviert sein sollte. Der eigentliche Nutzen der Anwendung ist die Unterstützung der Logistik und der Organisation des Unternehmens. Sie wird auch aus sicherheitstechnischen Gründen benötigt, z.B. bei der Überwachung der Fahrzeuge und von Personen.

Ich erhoffe mir, dass dieses Projekt in den Unternehmen, wo es zum Einsatz kommt, zu Verbesserungen der Wirtschaftlichkeit kommt und auch die Weiterentwicklung in Zusammenarbeit mit den Unternehmen und der inovisio communications GmbH erfolgt.

Literaturverzeichnis

[1] Himmelsrichtung – Wikipedia. https://de.wikipedia.org/wiki/Himmelsrichtung. Aufruf: 27.10.2016.

[2] Prof. Dr. Konrad Billwitz, Prof. Dr. Wolfgang Bricks, Dr. Bernd Raum. *Duden - Basiswissen Schule - Geografie*. paetec Gesellschaft für Bildung und Technik mbH, Berlin, 2002.

[3] Marc Jansen, Till Adams. *OpenLayers - Webentwicklung mit dynamischen Karten und Godaten*. Open Source Press, München, 2010.

[4] Prof. Dr. Hanno Lefmann. Vorlesung Theoretische Informatik II - SS 2007. https://www.tu-chemnitz.de/informatik/ThIS/downloads/courses/ss07/ti2/skript/script_ti2_2007.ps. Aufruf: 02.08.2016.

[5] Dr. Hubert Bossek, Prof. Dr. habil. Karlheinz Weber. *Duden - Basiswissen Schule - Mathematik Abitur*. DUDEN PAETEC GmbH, Berlin, 2007.

[6] Herleitung einer Formel für den Abstand zweier Punkte auf einer Kugel bei gegebenen Drehwinkeln und gegebenem Radius. http://www2.informatik.hu-berlin.de/~sanftleb/Kugelabstand.pdf. Aufruf: 13.09.2016.

[7] ICE 3: Deutschlands erster Zug für Tempo 300. https://www.bahn.de/p/view/service/zug/fahrzeuge/ice_3.shtml. Aufruf: 25.09.2016.

[8] Google Maps. http://maps.google.com. Aufruf: 12.08.2016.

[9] OpenLayers Simple Example – OpenStreetMap Wiki. http://wiki.openstreetmap.org/wiki/OpenLayers_Simple_Example. Aufruf: 02.08.2016.

[10] Google Maps API. http://maps.google.comhttp://maps.googleapis.com/maps/api/directions/json. Aufruf: 12.08.2016.

[11] Zeichen 209-20 – Wikimedia. https://upload.wikimedia.org/wikipedia/commons/6/6f/Zeichen_209-20.svg. Aufruf: 26.09.2016.

[12] Zeichen 101 – Wikimedia. https://upload.wikimedia.org/wikipedia/commons/0/02/Zeichen_101_-_Gefahrstelle%2C_StVO_1970.svg. Aufruf: 26.09.2016.

[13] Zeichen 123 – Wikimedia. https://upload.wikimedia.org/wikipedia/commons/3/3a/Zeichen_123.svg. Aufruf: 26.09.2016.

[14] Geofencing :: geofencing :: ITWissen.info. http://www.itwissen.info/definition/lexikon/Geofencing-geofencing.html. Aufruf: 25.08.2016.

[15] Punkt-in-Polygon-Test nach Jordan – Wikipedia. https://de.wikipedia.org/wiki/Punkt-in-Polygon-Test_nach_Jordan. Aufruf: 29.08.2016.

[16] Algorithmen der Computer-Geometrie. http://www.dbs.ifi.lmu.de/Lehre/GIS/SS2009/Skript/Kap-06.pdf. Aufruf: 31.08.2016.

[17] Dr. Marcel Heinz. Computergraphik I - 6. Mathematische Grundlagen. https://www.tu-chemnitz.de/informatik/GDV/lehre/lehrmaterial/skripte/571050/06_mathematische_grundlagen_teil1_1.pdf. Aufruf: 30.08.2016.

Selbstständigkeitserklärung

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keinem anderen Prüfer als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Chemnitz, den 05. Dezember 2016

_____

Martin Rein