



TECHNISCHE UNIVERSITÄT  
CHEMNITZ

Fakultät für Informatik

# MASTERARBEIT

Eine Methode für das Indoor-Routing auf  
Basis von OpenStreetMap-Daten

Vorgelegt von: Bettina Auschra  
[bettina.auschra@s2015.tu-chemnitz.de](mailto:bettina.auschra@s2015.tu-chemnitz.de)

Matrikelnummer: 421997

Studiengang: Informatik für Geistes- und Sozialwissenschaftler

Gutachter: Dr. Andreas Müller

Fachbetreuer: Dipl.-Ing. Thomas Graichen

Abgabedatum: 16.11.2018

# Inhalt

Abbildungsverzeichnis .....	iv
Aufgabenstellung.....	vi
1 Einleitung .....	1
2 Motivation .....	3
3 Grundlagen und Problembeschreibung .....	4
3.1 Framework.....	4
3.2 Tool zur Generierung von Wegenetzwerken.....	4
3.3 <i>OpenStreetMap</i> .....	5
3.3.1 OSM-Datenmodell.....	5
3.3.2 <i>Simple Indoor Tagging</i> .....	7
3.4 Mögliche Kriterien beim Indoor-Routing.....	8
3.4.1 Vom Outdoor-Routing übertragene Kriterien .....	8
3.4.2 Indoor-spezifische Routingkriterien .....	9
3.5 Aktueller Forschungsstand .....	10
3.5.1 Web-Applikation der Eidgenössischen Technischen Hochschule Lausanne (EFPL) .....	10
3.5.2 Web-Applikation der Universität Heidelberg.....	11
3.5.3 Mobile Applikation der Universität Potsdam .....	11
3.5.4 Mobile Applikation des Centro Universitario de Tecnología y Arte Digital (U-TAD) Madrid .....	11
3.5.5 Worboys und Yang: Automatische Generierung von Navigationsgraphen .	12
3.5.6 SIMON: Hybride Indoor-/Outdoor-Navigation.....	12
3.5.7 Mobile Applikation der Universität Bologna .....	13
3.6 Zusammenfassung und Ziele .....	13
4 Schaffung eines einheitlichen Schemas für die Beschreibung von Gebäuden.....	15
4.1 Eingänge .....	15
4.2 Räume und Korridore .....	16
4.3 Treppen und Fahrstühle .....	17
4.4 Verbindungen zwischen mehreren Etagen .....	18
4.5 Wege innerhalb von Gebäuden.....	19
5 Überarbeitung und Erweiterung des Tools zur Generierung von Wegenetzwerken ..	21
5.1 Hilfsfunktionen.....	21
5.1.1 Überprüfen, ob sich ein Punkt innerhalb des Raumes befindet.....	21
5.1.2 Überprüfen, ob sich ein Weg innerhalb des Raumes befindet .....	23
5.2 Die Klasse <i>Room</i> .....	23

5.2.1 Wege generieren auf Basis des <i>Straight Skeletons</i> .....	23
5.2.2 Wege vereinfachen .....	24
5.2.3 Direkte Verbindungen zwischen Türen suchen.....	25
5.2.4 Sich kreuzende Wege teilen .....	27
5.3 Die Klasse <i>Connection</i> .....	27
5.3.1 Wege über Treppen und Fahrstühle speichern .....	28
5.4 Die Klasse <i>Parser</i> .....	29
5.5 Funktionsweise des Tools nach Übernahme der Änderungen.....	29
5.6 Auswertung der generierten Wege .....	30
5.6.1 Durch <i>Straight Skeleton</i> generierte Wege .....	30
5.6.2 Durch <i>Door-to-Door</i> -Algorithmus generierte Wege.....	33
6 Auswahl und Anpassung einer Routingmaschine für den Indoorbereich .....	34
6.1 Erstellen eines Navigationsgraphen.....	35
6.1.1 Speichern der Etagenwerte .....	36
6.1.2 Hinzufügen von <i>Flag Encodern</i> .....	36
6.2 Suchanfrage starten.....	37
6.2.1 Kanten des Graphen filtern.....	37
6.3 Route berechnen .....	38
6.3.1 Hinzufügen der Etagenwerte zum Routingergebnis .....	38
6.4 Graphische Darstellung der Routen unter <i>GraphHopper</i> .....	41
6.4.1 <i>MiniGraphUI</i> .....	41
6.4.2 <i>MiniGraphUIIndoor</i> .....	43
6.4.3 <i>GHSserver</i> .....	45
6.4.4 Erweiterung von <i>GHSserver</i> für den Indoor-Bereich .....	47
7 Nutzertest.....	50
7.1 Zeichenteil .....	50
7.1.1 Zeichenaufgabe 1.....	52
7.1.2 Zeichenaufgabe 2.....	53
7.1.3 Zeichenaufgabe 3.....	54
7.1.4 Zeichenaufgaben 4 und 5.....	56
7.1.5 Zusammenfassung und Schlussfolgerungen.....	62
7.2 Befragung der Teilnehmer.....	64
7.2.1 Mögliche Anwendungsszenarien.....	64
7.2.2 Kriterien bei der Routenführung .....	65
7.2.3 Sonstige Erwartungen und Vorschläge .....	66
8 Fazit .....	67

9 Ausblick.....	69
Literaturverzeichnis .....	71
Anhang .....	74

# Abbildungsverzeichnis

1.1:	Die Heinrich-Zille-Straße in Chemnitz in einer <i>OSM</i> -Karte .....	6
1.2a:	Hydrant in der Heinrich-Zille-Straße als <i>Node</i> gespeichert.....	6
1.2b:	Heinrich-Zille-Straße als <i>Way</i> gespeichert.....	6
1.2c:	Fläche eines Gebäudes in der Heinrich-Zille-Straße als <i>Way</i> gespeichert.....	7
2:	Buslinie 82 in Chemnitz.....	7
3:	Eingänge im <i>OSM</i> -Format .....	16
4:	Raum und Korridor im <i>OSM</i> -Format gemäß <i>Simple Indoor Tagging</i> .....	16
5:	Multipolygon-Relation im <i>OSM</i> -Format .....	17
6:	Treppe und Fahrstuhl im <i>OSM</i> -Format.....	18
7:	Beispiel für eine Treppe mit Verzweigung .....	18
8:	Verbindungen zwischen mehreren Etagen im <i>OSM</i> -Format .....	19
9:	Weg über eine Etage im <i>OSM</i> -Format .....	19
10:	Wege über mehrere Etagen im <i>OSM</i> -Format.....	20
11:	Verdeutlichung der Strahlmethode .....	22
12:	Probleme bei der Strahlmethode .....	22
13:	Verdeutlichung der Kriterien zur Bestimmung, ob sich ein Weg innerhalb eines Raumes befindet.....	23
14:	Vergleich zwischen <i>Straight Skeletons</i> mit und ohne Übergabe der Barriere als Loch.....	24
15:	Funktionsweise von <i>simplify_ways</i> .....	25
16:	Vergleich zwischen Wegen, die von Tür zu Tür bzw. auf Mittellinien verlaufen .....	25
17:	Verschiedene Routen bei Vertauschung von Ausgangs- und Zielpunkt beim <i>Door-to-Door</i> -Algorithmus nach Liu und Zlatanova .....	26
18:	Sich kreuzende Wege (Ausschnitt aus generierten Wegen für das NHG) .....	27
19:	Funktionsweise des Tools zur automatischen Generierung von Wegenetzwerken .....	30
20:	Wege, die nicht mit Eingängen verbunden sind, und ein isolierter Weg .....	31
21:	Dicht beieinander verlaufende Wege, die zu Umwegen führen können.....	31
22:	<i>Straight Skeleton</i> bei der Übergabe von Löchern .....	32
23:	Generierte Wege nach dem Aufruf der Funktion <i>add_supplementary_ways</i> .....	33
24:	Generierte Wege mit und ohne <i>Door-to-Door</i> -Algorithmus .....	33
25:	<i>Pillar Nodes</i> und <i>Tower Nodes</i> .....	35
26:	Annäherungen von übergebenen Punkten bei der Suchanfrage.....	37
27:	Virtuelle Kante im Navigationsgraph .....	39
28:	Funktionsweise des Algorithmus zur Bestimmung der Etagenwerte der einzelnen Knoten des Pfades .....	40
29:	Navigationsgraph für Autos für die Region Sachsen unter <i>MiniGraphUI</i> .....	42
30:	Anzeige einer Route für Autos unter <i>MiniGraphUI</i> .....	42
31:	Darstellung des Navigationsgraphen für Fußgänger des NHGs unter <i>MiniGraphUI</i> .....	44
32:	Darstellung des Navigationsgraphen des NHGs für die drei implementierten <i>Flag Encoder</i> unter <i>MiniGraphUIIndoor</i> .....	44
33:	Darstellung einer Route vom ersten Obergeschoss ins Untergeschoss unter <i>MiniGraphUIIndoor</i> .....	45
34:	Darstellung der Region Sachsen unter <i>GHServer</i> .....	46
35:	Darstellung einer Route im Chemnitzer Stadtgebiet unter <i>GHServer</i> .....	47

36:	Darstellung einer Route zwischen dem Erdgeschoss und dem ersten Obergeschoss bei Vermeidung von Fahrstühlen .....	48
37:	Darstellung der Route zwischen den Punkten aus Abb. 33 bei Vermeidung von Treppen .....	49
38:	Bearbeitungen der ersten Zeichenaufgabe .....	53
39:	Durch <i>GraphHopper</i> berechnete Route für die erste Aufgabe .....	53
40:	Bearbeitungen der zweiten Zeichenaufgabe .....	
41:	Durch <i>GraphHopper</i> berechnete Routen für die zweite Aufgabe .....	54
42:	Bearbeitungen der dritten Zeichenaufgabe .....	55
43:	Durch <i>GraphHopper</i> berechnete Routen für die dritte Aufgabe .....	56
44.1:	Bearbeitungen der vierten Zeichenaufgabe.....	57
44.2:	Bearbeitungen der vierten Zeichenaufgabe.....	58
45:	Durch <i>GraphHopper</i> berechnete Routen für die vierte Aufgabe .....	59
46.1:	Bearbeitungen der fünften Zeichenaufgabe .....	60
46.2:	Bearbeitungen der fünften Zeichenaufgabe .....	61
47.1:	Durch <i>GraphHopper</i> berechnete Route für die fünfte Aufgabe .....	61
47.2	Durch <i>GraphHopper</i> berechnete Route für die fünfte Aufgabe .....	62

# Aufgabenstellung

Navigationssysteme sind heute aus dem Alltag vieler Nutzer nicht mehr wegzudenken und bieten eine erhebliche Unterstützung bei der Orientierung in unbekannten Gebieten und Städten. Neben der Unterstützung bei der Orientierung in Außenbereichen findet gegenwärtig auch die Innenraumnavigation immer mehr Anwendung. Populäre Dienste, wie „Google Maps“, bieten beispielsweise an Flughäfen oder Museen die dortigen Lagepläne integriert in die Kartendarstellung an und helfen ihren Nutzern dadurch ihre gewünschten Zielorte zu erreichen.

Leider sind diese populären Dienste proprietär sowie geschlossen und bieten daher nicht die Möglichkeit, die kartographierten Innenraumdaten für die weitere Verarbeitung, zum Beispiel für die Korrektur von Positionsdaten oder zur Berechnung von Routen, zu verwenden. Daher wurde im Rahmen verschiedener Forschungsprojekte an der Professur Schaltkreis- und Systementwurf ein Framework zur Verarbeitung von Innenraumdaten auf Basis von OpenStreetMap (OSM)-Karten entwickelt, das den Zugriff auf die Rohdaten der digitalen Karte ermöglicht und somit die Basis für weitere, neue Anwendungen bildet. Eine wichtige, bisher nicht betrachtete Komponente des Frameworks ist die Berechnung von Indoor-Routen aus vorliegenden Innenraumdaten. Die vorliegende Masterarbeit soll daher Methoden zur automatisierten Extraktion von begehbarer Pfaden aus bestehenden Indoor-OSM-Daten (Indoor-Flächen) sowie darauf aufbauend die Berechnung von Routen untersuchen. Basierend auf der Auswahl geeigneter Methoden soll eine der Problemstellung angemessene Methodik umgesetzt werden. Des Weiteren soll die Routing-Lösung die Berechnung von Routen für Menschen mit körperlicher Beeinträchtigung, wie bspw. Rollstuhlfahrer, berücksichtigen.

Zum Erreichen der Ziele sollen im Rahmen der Master-Arbeit folgende Teilaufgaben bearbeitet werden:

1. Literaturrecherche zu Verfahren und bestehenden Software-Bibliotheken für die (Indoor)-Routenberechnung
2. Auswahl eines geeigneten Verfahrens bzw. einer geeigneten Bibliothek nach Rücksprache mit den Betreuern
3. Identifikation und Bewertung der notwendigen Erweiterungen und Anpassungen der recherchierten Verfahren/Bibliotheken
4. Umsetzung der Erweiterungen sowie detaillierte Dokumentation dieser
5. Test und Evaluierung der Umsetzung mit verschiedenen OSM-Indoor-Datensätzen
6. Basierend auf der Evaluierung soll weiterhin geprüft werden, ob die bestehende OSM-Spezifikation zur Indoor-Kartographierung Schwächen hinsichtlich des Routings aufweist. Die ermittelten Schwächen sollen bewertet und durch Verbesserungsvorschläge ausgeräumt werden.

In der schriftlichen Arbeit sind die Grundlagen der verwendeten Verfahren ausführlich zu beschreiben und deren Einfluss auf die technische Umsetzung zu erläutern. Es wird ausdrücklich darauf hingewiesen, dass zur Lösung dieser Arbeit bestehende, quellenoffene Software eingesetzt und angepasst werden kann. Bei Verwendung solcher Software ist in der schriftlichen Arbeit deren Lizenz und Einsatzbedingungen (z.B. Software-Abhängigkeiten, unterstützte Betriebssysteme) zu nennen.

# 1 Einleitung

Navigationssysteme sind aus dem heutigen Leben kaum noch wegzudenken. Zahlreiche Anbieter von Navigationsanwendungen ermöglichen es ohne viel Aufwand, den Weg von A nach B herauszufinden und sich so in unbekannten Gegenden zu orientieren. Dabei gibt es Anwendungen, die auf die Bedürfnisse von bestimmten Benutzergruppen zugeschnitten sind. So ist es zum Beispiel häufig möglich, sich neben Routen für Autofahrer auch solche anzeigen zu lassen, die für Fahrradfahrer oder Fußgänger optimiert wurden.

Seit der Einführung der ersten standortbezogenen Dienste liegt der Forschungsschwerpunkt auf der Entwicklung von Navigationssystemen für den Außenbereich [FH17, S. 1]. Die Indoornavigation wurde aus mehreren Gründen lange als Forschungsthema vernachlässigt. Zum einen ist die Standortbestimmung innerhalb von Gebäuden deutlich komplizierter als außerhalb [TAK+05, S. 2]. Zum anderen gibt es innerhalb von Gebäuden keine vorgegebenen Wege. Während bei einem Outdoor-Routing auch die Fußgängerwege sich größtenteils an vorhandenen Straßen orientieren, muss im Indoor-Routing über Flächen navigiert werden. Des Weiteren muss bei der Darstellung von Gebäuden ein deutlich höherer Maßstab gewählt werden als bei der Darstellung von Straßennetzen. Durch die eventuelle Überlagerung mehrerer Etagen ist die topologische Struktur von Gebäuden außerdem deutlich komplexer [vgl. LLC+09, S. 382].

Seit einigen Jahren gewinnt die Indoornavigation jedoch zunehmend an Interesse [vgl. SSO08, S. 5]. Allein im Google Playstore ist eine Vielzahl an Anwendungen zu finden, die ein Indoor-Routing anbieten. Dabei handelt es sich jedoch in der Mehrzahl der Fälle um Applikationen, die für ein bestimmtes Gebäude, etwa für eine Universität oder ein Einkaufszentrum, entwickelt wurden. Dies hat den Nachteil, dass unklar ist, wie leicht sich die Routinglösungen auf andere Gebäude übertragen lassen. Hinzu kommt, dass die vorhandenen Anwendungen fast ausschließlich von kommerziellen Anbietern entwickelt wurden. Als Konsequenz wird für den Großteil der öffentlichen Gebäude noch kein Indoor-Routing angeboten.

Bei der Navigation im Außenbereich ist der Kartenanbieter *OpenStreetMap (OSM)* eine beliebte quelloffene Alternative zu kommerziellen Anbietern. Der Vorteil besteht darin, dass die Kartendaten jederzeit von allen Benutzern der Plattform angepasst werden können. So kann schnell auf strukturelle Veränderungen reagiert werden. Für die Darstellung

von Gebäudedaten gibt es bereits einen Vorschlag von der *OSM*-Community. Ein Routing innerhalb dieser Gebäude wird jedoch noch nicht angeboten. Die vorliegende Abschlussarbeit wird sich deshalb der Entwicklung einer Methode widmen, die das Indoor-Routing innerhalb von Gebäuden auf Basis von *OSM*-Daten ermöglicht.

Es kann zwar behauptet werden, dass die Unterstützung bei der Navigation innerhalb von Gebäuden weniger relevant ist als die Navigation außerhalb dieser [vgl. TAK+05, S.2]. So nimmt es für eine Person, die in einem Gebäude vom Weg abgekommen ist, durch die stärkere geographische Einschränkung normalerweise relativ wenig Zeit und Mühe in Anspruch, ihr Ziel trotzdem zu erreichen (ebd.). Dies trifft jedoch nicht auf sehr große und komplizierte Gebäude zu (ebd.). Für ältere Menschen oder Menschen mit körperlichen Einschränkungen ist es außerdem in besonderem Maße wichtig, einen möglichst kurzen beziehungsweise barrierefreien Weg zu finden (ebd.). Aus diesem Grund soll die zu entwickelnde Methode die Filterung von Wegen zulassen, die für körperlich Beeinträchtigte zugänglich sind.

Den Ausgangspunkt für diese Arbeit bilden ein zuvor abgeschlossenes Forschungsseminar und ein an der Professur Schaltkreis- und Systementwurf der TU Chemnitz entwickeltes Framework. Im Rahmen des Forschungsseminars wurde ein Tool entwickelt, mit dessen Hilfe aus Innenraumdaten im *OSM*-Format automatisch Wegenetzwerke generiert werden. Die vom Tool generierten Wege sollen die Datengrundlage für die zu entwickelnde Routingmethode bilden. Das erwähnte Framework stellt zurzeit Indoordaten im *OSM*-Format dar. Langfristig soll es möglich sein, die Routingmethode in das Framework zu integrieren. Die Methode soll dabei ausschließlich auf quelloffener Software basieren, sodass sie beliebig erweitert oder angepasst werden kann.

Im Folgenden wird zunächst genauer auf die Motivation für diese Abschlussarbeit eingegangen. Anschließend werden die Grundlagen, auf die diese Arbeit sich stützt, vorgestellt und die Problemstellung erläutert.

## 2 Motivation

Wie oben angeführt, kann ein Indoornavigationssystem körperlich beeinträchtigte Menschen dabei unterstützen, effizient an ihr Ziel zu gelangen. So kann sichergestellt werden, dass die jeweilige Person nicht durch eine Barriere daran gehindert wird, ihr Ziel zu erreichen. Aber auch Personen ohne körperliche Beeinträchtigung können von einem Indoornavigationssystem profitieren. So gibt es laut Fellner et al. [FHG17, S.1] Studien, die beweisen, dass Menschen die Orientierung innerhalb von Gebäuden schneller verlieren als außerhalb dieser. Eine Indoornavigation könnte dem entgegenwirken. Die Forschung ist sich zudem darüber einig, dass eine Indoornavigation vor allem in komplexen Gebäuden wie Krankenhäusern, Universitäten, großen Einkaufszentren und Flughäfen sinnvoll ist (vgl. [FHG17, S.1; TAK+05, S. 2]).

Grundsätzlich erleichtert ein Navigationssystem Menschen das Zurechtfinden an unbekannten Orten. Innerhalb von Gebäuden ist dies vor allem wichtig, wenn Zeitdruck herrscht. Fallah et al. [FAB+13, S.22] illustrieren dies mit dem Beispiel eines Sanitäter- oder Feuerwehreinsatzes, bei dem die Rettungskräfte ihr Ziel so schnell wie möglich erreichen müssen. Bei letzterem kann die Orientierung durch Rauch zusätzlich erschwert werden, sodass eine Navigationsunterstützung in besonderem Maße sinnvoll erscheint (ebd.).

Alle Menschen sollten dieselben Chancen haben, sich in einem unbekannten Gebäude zurechtzufinden. Ein Indoor-Routenplaner ist ein erster Schritt in diese Richtung, weil er durch die Möglichkeit des Vorausplanens, das Vermeiden von individuellen Hindernissen zulässt. Der Vorteil eines Indoor-Routenplaners auf Basis von quelloffener Software ist, dass er jederzeit spezifischen Bedürfnissen angepasst werden kann.

## 3 Grundlagen und Problembeschreibung

Wie zuvor erwähnt, bilden ein im Forschungsseminar entwickeltes Tool zur Generierung von Wegenetzwerken innerhalb von Gebäuden und ein Framework der Professur Schaltkreis- und Systementwurf der TU Chemnitz die Grundlagen dieser Arbeit. Beide werden im Folgenden kurz vorgestellt. Des Weiteren wird auf *OpenStreetMap* und das *OSM*-Format eingegangen, da sich die gesamte Arbeit darauf stützt. Ein weiterer Abschnitt behandelt Kriterien, die bei einem Indoor-Routing beachtet werden können. Außerdem wird auf den aktuellen Forschungsstand eingegangen. Abschließend wird erläutert, welche Ziele sich daraus für diese Abschlussarbeit ableiten lassen.

### 3.1 Framework

Das in Java geschriebene Framework ermöglicht die Darstellung von Indoor- und Outdoordaten im *OpenStreetMap*-Format. Im Fall von Indoordaten wird für jede Etage ein zweidimensionaler Plan generiert. Mittels einer Anzahl von Buttons kann der Benutzer<sup>1</sup> zwischen den Etagenansichten wechseln. Diese Funktion, mit der die Etagen eines Gebäudes über Buttons gefiltert werden können, wird im Folgenden als Etagenselektor bezeichnet. Da es bereits quelloffene Routingmaschinen für den Außenbereich gibt, soll im Rahmen dieser Arbeit eine ausgewählt werden, die für das Indoor-Routing angepasst und später in das Framework integriert werden kann.

### 3.2 Tool zur Generierung von Wegenetzwerken

Mit Hilfe des in Python 3 geschriebenen Tools werden Gebäudedaten im *OSM*-Format eingelesen. Aus den eingelesenen Daten generiert das Tool automatisch ein Wegenetzwerk für das entsprechende Gebäude. Die gefundenen Wege werden anschließend in einer Datei im *OSM*-Format gespeichert. Genaueres zu der grundlegenden Funktionsweise des Tools kann in der Forschungsseminararbeit [Aus18] nachgelesen werden.

Im Rahmen des Forschungsseminars wurde noch nicht auf mehrstöckige Gebäude und die daraus resultierenden Verbindungen zwischen mehreren Etagen eingegangen. Aus

---

<sup>1</sup> Aus Gründen der besseren Lesbarkeit werden im Folgenden nur männliche Personenbezeichnungen verwendet. Es sind aber grundsätzlich immer alle Geschlechter gemeint.

diesem Grund muss das Tool in der vorliegenden Abschlussarbeit um diese Funktion erweitert werden. Des Weiteren wurden im Ausblick des Forschungsseminars einige Änderungen vorgeschlagen, auf die in dieser Arbeit ebenfalls eingegangen werden soll.

### 3.3 OpenStreetMap

*OpenStreetMap* ist ein Projekt, das 2004 ins Leben gerufen wurde [OSMa]. Derzeit hat das Projekt mehr als fünf Millionen registrierte Benutzer [OSMb]. Unter <https://www.openstreetmap.org/> kann eine skalierbare Weltkarte abgerufen werden. Die kartographischen Daten dieser Karte können von registrierten Benutzern beliebig bearbeitet und erweitert werden. Das Kartenmaterial darf frei weiterverwendet werden [OSMc]. Die Website bietet auch ein Routing an, welches jedoch auf Ziele im Außenbereich beschränkt ist.

Zipf et al. [ZMR+16, S.326] zufolge ist das Kartenmaterial von *OSM* genauer und aktueller als das von kommerziellen Anbietern, weil mehr Menschen dazu beitragen. Als Beispiel führen sie an, dass ein auf *OSM* basierender Anbieter früher als Google Maps Fußgänger- und Fahrradnavigation anbot.

Das *OSM*-Kartenmaterial wird im XML-Format beschrieben. Die einzelnen Elemente werden dabei durch das *OSM*-Datenmodell festgelegt. Beim *Simple Indoor Tagging* handelt es sich um eine Anwendung des *OSM*-Datenmodells zur Beschreibung von Gebäuden (vgl. [OSMi]). Das *OSM*-Modell und das *Simple Indoor Tagging* werden in den folgenden Abschnitten überblicksweise vorgestellt.

#### 3.3.1 OSM-Datenmodell

Mit Hilfe des *OSM*-Datenmodells können beliebige Strukturen, wie Straßen, Plätze, Gebäude oder wichtige Punkte dargestellt werden (vgl. Abb. 1.1). Die Grundelemente im *OSM*-Datenmodell bilden *Nodes*<sup>2</sup>, *Ways* und Relationen [OSMd]. Diese Begrifflichkeiten werden im Folgenden genauer erläutert. Jedes der Grundelemente besitzt eine innerhalb des Elementtyps eindeutige ID und kann beliebig viele *Tags* erhalten. *Tags* beschreiben als Schlüssel-Wert-Paare das jeweilige Element genauer ([OSMe], vgl. Abb. 1.2a-c).

---

<sup>2</sup> Für die Elemente, für die es keine eindeutige Übersetzung innerhalb der *OpenStreetMap*-Community gibt, werden die englischen Ausdrücke verwendet.



Abbildung 1.1: Die Heinrich-Zille-Straße in Chemnitz in einer OSM-Karte

Ein *Node* ist ein einzelner Punkt im Raum, der durch einen Längen- und einen Breitengrad definiert wird [OSMg]. Optional kann zusätzlich eine Höhe für den Punkt vergeben werden. *Nodes* können auch die Positionen von wichtigen Strukturen kennzeichnen (vgl. Abb. 1.2a).

```
<node id="4874455650" lat="50.8425523" lon="12.9288578">
  <tag k="emergency" v="fire_hydrant"/>
  <tag k="fire_hydrant:diameter" v="200"/>
  <tag k="fire_hydrant:position" v="Lane"/>
  <tag k="fire_hydrant:type" v="underground"/>
  <tag k="ref" v="14179"/>
</node>
```

Abbildung 1.2a: Hydrant in der Heinrich-Zille-Straße als Node gespeichert (vgl. Abb. 1.1)

Ein *Way* besteht aus einer geordneten Liste von durch die IDs referenzierten *Nodes* [OSMh]. Ein *Way* kann einen Weg, etwa eine Straße oder einen Pfad (vgl. Abb. 1.2b), oder eine Fläche, zum Beispiel die eines Platzes oder eines Gebäudes (vgl. Abb. 1.2c), darstellen. Im Fall von Flächen müssen der erste und der letzte referenzierte *Node* dieselben sein.

```
<way id="30747527" >
  <nd ref="340049751"/>
  <nd ref="340049758"/>
  <tag k="bicycle" v="yes"/>
  <tag k="foot" v="yes"/>
  <tag k="highway" v="residential"/>
  <tag k="maxspeed" v="30"/>
  <tag k="name" v="Heinrich-Zille-Straße"/>
</way>
```

Abbildung 1.2b: Heinrich-Zille-Straße als Way gespeichert (vgl. Abb. 1.1)

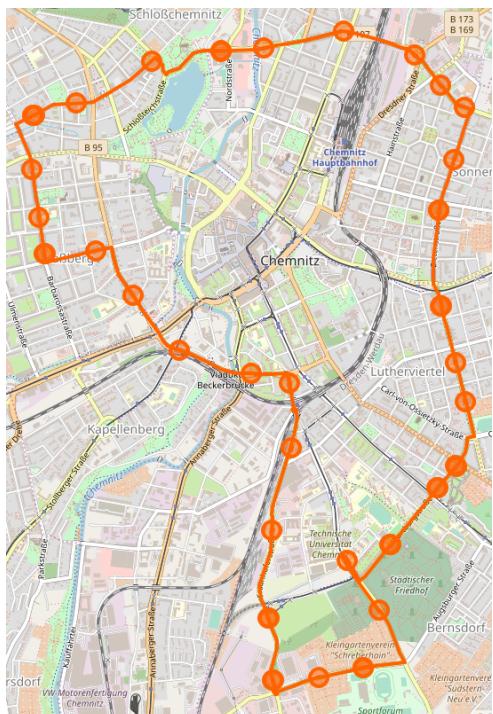
```

<way id="130112855">
  <nd ref="1434097394"/>
  <nd ref="1434097409"/>
  <nd ref="1434097428"/>
  <nd ref="1434097421"/>
  <nd ref="1434097394"/>
  <tag k="building" v="yes"/>
</way>

```

Abbildung 1.2c: Fläche eines Gebäudes in der Heinrich-Zille-Straße als Way gespeichert (vgl. Abb. 1.1)

Relationen werden benötigt, um Zusammenhänge zwischen Objekten darzustellen [OSMF]. Ein Beispiel sind Buslinien, die sich aus mehreren Bushaltestellen (*Nodes*) und Straßen (*Ways*) zusammensetzen (vgl. Abb. 2). Die einzelnen Objekte der Relation werden als *Members* bezeichnet. Eine Relation kann *Nodes*, *Ways* oder auch weitere Relationen als *Member* enthalten. Jede Relation muss mindestens ein *Tag* besitzen.



```

<relation id="7796708">
  <member type="node" ref="5239435218"
    role="stop"/>
  <member type="way" ref="545750524"
    role="platform"/>
  <member type="node" ref="3643796510"
    role="stop"/>
  <member type="way" ref="359717699"
    role="platform"/>
  (...)<br/>
  <member type="way" ref="359717645"/>
  <tag k="name" v="Bus 82: Ringbus"/>
  <tag k="network" v="VMS"/>
  <tag k="note" v="Ringbuslinie mit
    Fahrtroute im Uhrzeigersinn"/>
  <tag k="operator" v="CVAG"/>
  <tag k="public_transport:version"
    v="2"/>
  <tag k="ref" v="82"/>
  <tag k="route" v="bus"/>
  <tag k="type" v="route"/>
</relation>

```

Abbildung 2: Buslinie 82 in Chemnitz  
Links: hervorgehoben in einer OSM-Karte; rechts: Ausschnitt aus der Beschreibung als Relation

### 3.3.2 Simple Indoor Tagging

Beim *Simple Indoor Tagging* handelt es sich um ein Schema zur einheitlichen Beschreibung von Gebäudeinneren im *OSM*-Format, auf das sich die *OSM*-Community geeinigt hat. Es legt fest, welche *Tags* für welche Gebäudeteile vergeben werden. Das *Simple Indoor Tagging* geht allerdings noch nicht auf alle Aspekte innerhalb eines Gebäudes ein.

So gibt es zum Beispiel keine einheitliche Regelung dazu, wie Türen dargestellt werden sollen<sup>3</sup>. Teilweise sind die derzeitigen Vorschläge für die Darstellung von Gebäudeteilen gemäß dem *Simple Indoor Tagging* außerdem in sich nicht schlüssig. Deshalb muss zunächst ein einheitlicher Standard entwickelt werden, auf den das Tool zur Wegenetzgenerierung und die Routingmaschine sich anschließend stützen können. Dies wird auch Bestandteil dieser Arbeit sein. Als Datengrundlage wird das Neue Hörsaalgebäude (NHG) der TU Chemnitz dienen, das bereits gemäß dem *Simple Indoor Tagging* im OSM-Format beschrieben wurde.

### 3.4 Mögliche Kriterien beim Indoor-Routing

Dieser Abschnitt widmet sich den möglichen Kriterien, die bei der Routenplanung innerhalb von Gebäuden beachtet werden können. Zunächst wird auf Kriterien aus dem Outdoor-Bereich eingegangen, die sich auf den Indoor-Bereich übertragen lassen. Anschließend werden Kriterien, die beim Routing im Indoor-Bereich neu hinzukommen, beleuchtet.

#### 3.4.1 Vom Outdoor-Routing übertragene Kriterien

Beim Outdoor-Routing spielt die Art der Straße oder des Weges eine große Rolle. So muss etwa gewährleistet werden, dass Fußgänger nicht über Autobahnen und Autos nicht durch Fußgängerzonen navigiert werden. Auch innerhalb von Gebäuden gibt es Bereiche, die nur von bestimmten Personen betreten werden können. Dies trifft zum Beispiel auf Universitätsteile zu, die nur für Angehörige einer bestimmten Fakultät zugänglich sind. Andere Gebäude oder Gebäudeabschnitte wiederum können nicht durchgängig betreten werden, weil sie an Öffnungszeiten gebunden sind.

Bei einem Outdoor-Routenplaner, der auf die Bedürfnisse von Rollstuhlfahrern ausgerichtet ist, muss die Art des Rollstuhls berücksichtigt werden [BFB+06, S. 72]. So kann es für die Routenplanung entscheidend sein, ob es sich um einen motorisierten oder manuell betriebenen Rollstuhl handelt, da manche Steigungen mit ersterem besser zu überwinden sind [BFB+06, S. 72]. Ebenso relevant kann es sein, ob die Person im Rollstuhl Hilfe von einer weiteren Person beansprucht [Nei15, S. 188]. Dies sollte auch bei einem

---

<sup>3</sup> Stand: 07.08.2018

Indoor-Routenplaner berücksichtigt werden, etwa wenn sich innerhalb des Gebäudes eine Rampe befindet.

Die Oberflächenbeschaffenheit der Wege im Außenbereich spielt eine ausschlaggebende Rolle. So können Wege über Kopfsteinpflaster oder Kies Probleme für Rollstuhlfahrer darstellen [BFB+06, S. 72]. Im Rahmen dieser Arbeit wird davon ausgegangen, dass es innerhalb eines Gebäudes in Bezug auf die Oberflächenbeschaffenheit normalerweise wenig Varianz und dadurch kaum alternative Routen gibt. Eine Ausnahme bilden Änderungen in der Oberflächenbeschaffenheit, die durch Blindenleitsysteme zustande kommen. Da es innerhalb des NHGs keine Blindenleitsysteme gibt, kann dieses Kriterium in dieser Arbeit nicht genauer untersucht werden.

Sowohl für Personen im Rollstuhl als auch für Blinde oder Sehbehinderte ist außerdem die Breite des Bürgersteigs von Bedeutung [Nei15, S. 195; KDO10, S. 2225]. Übertragen auf Gebäude müsste daher auch auf die Breite von Türen oder Korridoren eingegangen werden. Es würde allerdings den Umfang dieser Arbeit sprengen, auf dieses Kriterium einzugehen. Nach Möglichkeit sollte es deshalb in zukünftigen Forschungsarbeiten genauer beleuchtet werden.

### 3.4.2 Indoor-spezifische Routingkriterien

Bei der Entwicklung von Navigationssystemen zur Nutzung innerhalb von Gebäuden gibt es Umstände zu beachten, die bei der Navigation in Außenbereichen keine Rolle spielen. Der offensichtlichste Unterschied ist vermutlich, dass Gebäude in den meisten Fällen aus mehreren Stockwerken bestehen. Bei der Erstellung von Routen innerhalb eines Gebäudes muss gegebenenfalls also auch auf Verbindungen zwischen mehreren Etagen eingegangen werden. Dabei muss beachtet werden, um welche Art von Verbindung es sich handelt. So sollte einem Rollstuhlfahrer kein Weg über Treppen vorgeschlagen werden. Andere Zielgruppen könnten dagegen eine Routenführung über Treppen bevorzugen.

Beale et al. [BFB+06, S. 69] fordern, auch die jeweilige Wartezeit auf einen Fahrstuhl zu berücksichtigen. Die Anzahl der Personen, die einen Fahrstuhl benutzen, und die Anzahl der Etagen, auf der dieser anhalten wird, sind nur zwei mögliche Faktoren, die die Wartezeit beeinflussen. Beide können nicht vorhergesesehen oder beeinflusst werden. Aus diesen Gründen wird dieses Kriterium als nicht messbar erachtet und ignoriert.

Lyardet et al. [LSA08, S. 306] schlagen vor, dass bei einer Indoor-Routenplanung, die die individuellen Bedürfnisse der Nutzer berücksichtigt, auch auf die Temperatur und die Helligkeit innerhalb der Räume eingegangen werden sollte. Da davon ausgegangen wird, dass es innerhalb eines Gebäudes normalerweise keine gravierenden Temperaturunterschiede gibt und die begehbarer Teile gemeinhin beleuchtet sind, werden diese Kriterien ebenfalls als unwichtig eingestuft. [LSA08, S. 306] fordern ebenfalls, die Anzahl an Menschen, die sich in einem Gebäudeteil befinden, zu berücksichtigen. Da sich diese innerhalb von kürzester Zeit ändern kann, wird sie auch als nicht messbares Kriterium betrachtet.

### 3.5 Aktueller Forschungsstand

Dieses Kapitel widmet sich bestehenden Indoor-Routing-Lösungen. Es wird auf Indoor-Routing-Lösungen eingegangen, die in der Forschungsliteratur vorgestellt wurden. Beschränkt wird sich dabei auf Arbeiten, die sich auf *OSM* stützen oder auf Kriterien eingehen, die für körperlich beeinträchtigte Menschen relevant sind.

#### 3.5.1 Web-Applikation der Eidgenössischen Technischen Hochschule Lausanne (EPFL)

Büchel & Gilliéron [BG04] präsentierten 2004 einen Indoor-Routenplaner für den Campus der EPFL. Der Routenplaner ist in Python geschrieben und greift auf eine PostgreSQL/PostGIS-Datenbank zu [BG04, S. 665]. Die Autoren weisen darauf hin, dass bei Berechnung der Routen das individuelle Profil des Benutzers berücksichtigt wird [BG04, S. 666]. So würden etwa für Rollstuhlfahrer Wege über Treppen ausgeschlossen. Außerdem werde die Zugangsberechtigung des Benutzers zu den durchquerten Gebäude-Teilen gewährleistet. Bei dem Plan, der unter <https://plan.epfl.ch> abrufbar ist, konnte jedoch keine Möglichkeit gefunden werden, solch ein Benutzerprofil anzulegen.

Beim Routing kann der Benutzer entweder eine Raumnummer oder den Namen einer Person, die ein Büro belegt, als Start oder Ziel eingeben. Dabei werden dem Benutzer auch Orte und Namen vorgeschlagen. Das Gelände um den Campus herum wird mit den Daten der *OSM*-Community dargestellt. Ein Routing außerhalb des Campus ist jedoch nicht möglich. Es konnten keine Informationen dazu gefunden werden, wie das Wege-Netzwerk innerhalb des Campus erstellt wurde.

### 3.5.2 Web-Applikation der Universität Heidelberg

Goetz und Zipf [GZ12] von der Universität Heidelberg stellten 2012 den *OSM*-Modellierungsvorschlag *IndoorOSM* für die Beschreibung von Gebäuden vor. Dabei entwickelten sie auch eine Web-Applikation, in der beispielhaft ein Gebäude dargestellt wurde, das mithilfe von *IndoorOSM* modelliert wurde [GZ12, S. 33]. Auch ein Routing innerhalb des Gebäudes und gegebenenfalls über mehrere Etagen hinweg wurde angeboten.

Der Modellierungsvorschlag *IndoorOSM* wurde inzwischen von der *OSM*-Community zugunsten des *Simple Indoor Taggings* verworfen (vgl. [OSMj]). Die Webseite der Web-Applikation ist zwar noch unter <http://indoorOSM.uni-hd.de> zu erreichen, es werden jedoch keine Gebäudedaten oder Routen mehr dargestellt.

### 3.5.3 Mobile Applikation der Universität Potsdam

An der Universität Potsdam wurde eine Android-Applikation für eine barrierefreie Navigation in den Gebäuden der drei Campus entwickelt [FMS+14]. Den Autoren zufolge kann der Benutzer ein individuelles Profil anlegen. In diesem soll festgehalten werden, ob und gegebenenfalls was für eine Einschränkung vorliegt und ob Treppen oder Fahrräder bei der Routenplanung miteinbezogen werden dürfen [FMS+14, S. 2406]. Benutzer mit Sehbehinderungen haben außerdem die Möglichkeit, das Farbschema der App an ihre Bedürfnisse anzupassen (ebd.). Auch eine Bedienung der App mittels Sprachsteuerung soll möglich sein [FMS+14, S. 2402]. Die App stellt den Indoor- und den Outdoorbereich im *OSM*-Format dar [FMS+14, S. 2405]. Das Format der Indoordaten beruht dabei ebenfalls auf dem veralteten Modellierungsvorschlag *IndoorOSM* [FMS+14, S. 2404]. Für genauere Informationen zu der Barrierefreiheit innerhalb der Universitätsgebäude greift die App laut Aussagen der Autoren auf eine dafür angelegte MySQL-Datenbank zu (ebd.). Die App konnte online nicht gefunden werden. Auch in diesem Fall ist nicht bekannt, wie das zugrundeliegende Wegenetzwerk erstellt wurde.

### 3.5.4 Mobile Applikation des Centro Universitario de Tecnología y Arte Digital (U-TAD) Madrid

Amat et al. [AFA+14] entwickelten eine mobile Indoor-Routing-Lösung für ein Gebäude der U-TAD-Universität in Madrid. Diese basiert auf *OSM* und einer NoSQL-Datenbank [AFA+14, S. 1]. Laut Aussagen der Autoren wurde das Gebäude wie an den Universitäten in Heidelberg und Potsdam auf der Grundlage von *IndoorOSM* modelliert (ebd.). Das

Wegenetzwerk für das Routing innerhalb des Gebäudes wurde manuell erstellt [AFA+14, S. 3]. Auch diese Applikation wurde online nicht gefunden.

### 3.5.5 Worboys und Yang: Automatische Generierung von Navigationsgraphen

Worboys und Yang [WY15] entwickelten ein Verfahren zur automatischen Generierung von Navigationsgraphen innerhalb von Gebäuden. Dieses wurde bereits im Rahmen des Forschungsseminars genauer vorgestellt. Die Autoren modellierten ein Gebäude der University of Maine gemäß dem *IndoorOSM*-Schema. Ob das automatisch generierte Wegenetzwerk auch für ein Routing verwendet wurde, konnte nicht herausgefunden werden.

### 3.5.6 SIMON: Hybride Indoor-/Outdoor-Navigation

Wagner et al. [WFH+17] von der Universität Duisburg-Essen entwarfen im Rahmen des EU-Forschungsprojekts SIMON<sup>4</sup> eine Applikation, die ein übergreifendes Routing zwischen Innen- und Außenbereich ermöglicht. Die Applikation unterstützt verschiedene Routingprofile, die die Bedürfnisse verschiedener Benutzergruppen mit Mobilität einschränkungen berücksichtigen [WFH+17, S. 39]. Auch hier wird die Außenumgebung mit den Daten von *OSM* dargestellt [WFH+17, S. 36]. Für die Darstellung der Gebäude daten wurde jedoch ein eigenes Format entwickelt [WFH+17, S. 37].

Bei der Erstellung des Wegenetzwerks werden zunächst die wichtigsten Wege per Hand eingezeichnet [WFH+17, S. 38]. Anschließend werden die einzelnen Punkte, die den Raum oder Objekte innerhalb des Raumes darstellen, automatisch paarweise untersucht. Befindet sich eine Verbindung zwischen zwei Punkten innerhalb des Raumes, wird das Wegenetzwerk um diese ergänzt. Die Autoren entschieden sich gegen einen vollkommen automatischen Ansatz, um Wege, die direkt an den Wänden entlanglaufen, zu vermeiden. Des Weiteren erläutern sie, dass bei einer rein automatischen Generierung der Wegenetzwerke Wege, die nur in eine Richtung zulässig sind, nicht berücksichtigt werden könnten.

---

<sup>4</sup> Siehe <http://simon-project.eu/project/>

### 3.5.7 Mobile Applikation der Universität Bologna

Delnevo et al. stellten Anfang des Jahres 2018 eine mobile Applikation für die Navigation auf den verschiedenen Campus der Universität Bologna vor. Die Anwendung richtet sich an Studierende und wurde explizit auch für solche mit Behinderungen entwickelt [DMV+, S. 1]. Innerhalb der Applikation können die Benutzer ein individuelles Profil anlegen, das es zum Beispiel ermöglicht, eine Route ausgegeben zu bekommen, die nur über rollstuhlgerechte Fahrstühle führt [DMV+18, S. 5]. Andersherum können Wege ausgeschlossen werden, die über Fahrstühle führen, was sinnvoll für Blinde sein kann, die auf einen Blindenhund angewiesen sind (ebd.). Zum Zeitpunkt des Verfassens dieser Arbeit war die Applikation nicht online zu finden.

## 3.6 Zusammenfassung und Ziele

Zusammenfassend kann festgestellt werden, dass die meisten bestehenden Indoor-Routing-Lösungen jeweils nur für ein bestimmtes Gebäude oder einen Gebäudekomplex entwickelt wurden. Da von keiner der oben beschriebenen Anwendungen der Quellcode frei verfügbar ist, kann nicht überprüft werden, wie leicht sich die bestehenden Lösungen auf andere Gebäude übertragen lassen. Hinzu kommt, dass sich diese Anwendungen bei der Darstellung der Gebäudedaten auf das *Simple Indoor Tagging* stützt. Die Anwendungen, welche Gebäudedaten im *OpenStreetMap*-Format darstellen, stützen sich auf den veralteten Modellierungsvorschlag *IndoorOSM*.

[WFH+17] und [WY15] schlagen als einzige einen Ansatz zur zumindest teilweisen automatischen Generierung von Wegenetzwerken vor. Dies bedeutet, dass diese Arbeit nach bestem Wissen die erste ist, die eine automatische Generierung von Wegenetzwerken mit einem Routing verbindet und sich dabei auf den neuen Modellierungsstandard *Simple Indoor Tagging* stützt.

Folgende Ziele wurden für diese Masterarbeit gesetzt:

1. Es soll ein einheitliches OSM-Format für die Darstellung von Gebäudedaten, die für ein Indoor-Routing relevant sind, geschaffen werden. Dieses soll auf dem *Simple Indoor Tagging* basieren.
2. Das bestehende Tool zur automatischen Wegenetzwerksgenerierung soll optimiert werden, insbesondere in Bezug auf die Generierung von Wegen über mehrere Etagen.

3. Aus bestehenden quelloffenen Routingsoftwares soll eine ausgewählt werden und dem Routing innerhalb von Gebäuden angepasst werden. Bei der Berechnung von Routen soll auch auf die Bedürfnisse verschiedener Nutzergruppen eingegangen werden. Der Fokus soll dabei zunächst auf den Bedürfnissen von körperlich Beeinträchtigten liegen.
4. Die vom Tool generierten Wegenetzwerke sowie die von der Routingsoftware berechneten Routen sollen ausgewertet werden.

## 4 Schaffung eines einheitlichen Schemas für die Beschreibung von Gebäuden

Die Struktur der *OSM*-Daten, mit denen die im Rahmen dieser Masterarbeit entwickelten Algorithmen getestet wurden, orientiert sich am *Simple Indoor Tagging*. Dieses Schema wurde teilweise im Rahmen des Forschungsseminars und der Masterarbeit angepasst. Im Folgenden werden die Elemente von Gebäuden vorgestellt, die für die Generierung von Wegenetzwerken in dieser Arbeit relevant sind. Des Weiteren wird auf die Beschreibung der Wege selbst durch Gebäude eingegangen. Änderungen und Erweiterungen des *Simple Indoor Taggings* sowie die Gründe dafür werden explizit erwähnt. Die erwähnten Änderungen wurden in das Kartenmaterial des NHGs übernommen.

Mit Ausnahme der Relationen müssen alle Elemente, die hier vorgestellt werden, ein *Tag* besitzen, das sich aus dem Schlüssel „level“ und dem Wert der Etage, auf dem diese sich befinden, zusammensetzt. Auf diesen Umstand wird deshalb nicht erneut eingegangen.

### 4.1 Eingänge

Für die Speicherung von Eingängen gibt es zum Zeitpunkt des Schreibens gemäß dem *Simple Indoor Tagging* noch keine einheitliche Regelung. In den Indoordaten, die in dieser Arbeit verwendet werden, werden sie als *Nodes* oder als Flächen in Form von *Ways*<sup>5</sup> gespeichert (vgl. Abb. 3). Handelt es sich um einen Eingang, der vom Außengelände in das betreffende Gebäude führt, erhält er ein *Tag* mit dem Schlüssel „entrance“. Als Wert wird entweder „yes“ oder die Art des Eingangs hinzugefügt. Ein Eingang, der einen Raum oder Korridor mit einem anderen Raum oder Korridor innerhalb eines Gebäudes verbindet, erhält er ein *Tag* mit dem Schlüssel „door“. Besitzt der Eingang keine physische Tür, erhält das *Tag* den Wert „no“, anderenfalls den Wert „yes“ oder die Art der Tür. Optional kann bei beiden Arten von Eingängen noch ein *Tag* mit dem Schlüssel „access“ hinzugefügt werden, dessen Wert aussagt, ob der Eingang öffentlich zugänglich ist oder nicht.

---

<sup>5</sup> Da bei der Berechnung von Wegen für Eingänge, die als *Ways* gespeichert wurden, erst der Schwerpunkt der Fläche gebildet werden muss, wird die Speicherung von Eingängen als *Nodes* bevorzugt.

```

<node id='1' lat='52.08156819998' lon='62.99240761514'>
  <tag k='access' v='private' />
  <tag k='door' v='yes' />
  <tag k='level' v='1' />
</node>

<node id='2' lat='46.08560764712' lon='56.85287095327'>
  <tag k='door' v='no' />
  <tag k='level' v='1' />
</node>

<way id='1'>
  <nd ref='3' />
  <nd ref='4' />
  <nd ref='5' />
  <nd ref='6' />
  <nd ref='3' />
  <tag k='entrance' v='yes' />
  <tag k='level' v='0' />
</way>

```

Abbildung 3: Eingänge im OSM-Format

Von oben nach unten: Tür innerhalb des Gebäudes mit privatem Zugang, Eingang ohne physische Tür, Eingang vom Außengelände ins Gebäude ohne Zugangsbeschränkung

## 4.2 Räume und Korridore

Räume und Korridore werden als *Ways*, die Flächen darstellen, gespeichert (vgl. Abb. 4). Die *Ways* müssen ein Tag „*indoor=room*“ beziehungsweise „*indoor=corridor*“ besitzen. Mithilfe von zusätzlichen *Tags* können weitere Eigenschaften, etwa die Bezeichnung eines Raumes, vergeben werden.

<pre> &lt;way id='1'&gt;   &lt;nd ref='1' /&gt;   &lt;nd ref='2' /&gt;   &lt;nd ref='3' /&gt;   &lt;nd ref='4' /&gt;   &lt;nd ref='1' /&gt;   &lt;tag k='indoor' v='room' /&gt;   &lt;tag k='level' v='1' /&gt; &lt;/way&gt; </pre>	<pre> &lt;way id='2'&gt;   &lt;nd ref='5' /&gt;   &lt;nd ref='6' /&gt;   &lt;nd ref='7' /&gt;   &lt;nd ref='8' /&gt;   &lt;nd ref='5' /&gt;   &lt;tag k='indoor' v='corridor' /&gt;   &lt;tag k='level' v='0' /&gt; &lt;/way&gt; </pre>
---	---

Abbildung 4: Raum (links) und Korridor im OSM-Format gemäß Simple Indoor Tagging

Befinden sich innerhalb eines Raums beziehungsweise Korridors eine oder mehrere Barrieren, wird eine sogenannte Multipolygon-Relation angelegt (vgl. Abb. 5). Bei Barrieren kann es sich beispielsweise um Säulen, Zwischenwände oder auch um einen Raum handeln, der von einem weiteren Raum oder einem Korridor umgeben ist. Eine Multipolygon-Relation erhält mindestens zwei *Ways* als *Members*, die die Flächen des Raums und einer oder mehrerer Barrieren darstellen. Das *Member*, das den Raum abbildet, erhält

dabei das Schlüssel-Wert-Paar „role=outer“ und jede einzelne Barriere das Schlüssel-Wert-Paar „role=inner“. Um eine Multipolygon-Relation als solche zu kennzeichnen, erhält sie das Schlüssel-Wert-Paar „type=multipolygon“.

```

<relation id='1'>
  <member type='way' ref='1' role='inner' />
  <member type='way' ref='2' role='inner' />
  <member type='way' ref='3' role='inner' />
  <member type='way' ref='4' role='outer' />
  <tag k='type' v='multipolygon' />
</way>
```

Abbildung 5: Multipolygon-Relation im OSM-Format

### 4.3 Treppen und Fahrstühle

Die Flächen von Treppen und Fahrstühlen werden für jede Etage einzeln gespeichert. Sie werden ebenfalls als *Ways* erfasst und erhalten die *Tags* „indoor=area“ und „stairs=yes“ beziehungsweise „elevator=yes“ (vgl. Abb. 6). Die Zugangspunkte zu den Korridoren, an die die Treppe beziehungsweise der Fahrstuhl grenzen, werden laut *Simple Indoor Tagging* als Eingänge mit dem Schlüssel „door“ gekennzeichnet. Auf die Verbindung zu Räumen wird in der Beschreibung vom *Simple Indoor Tagging* nicht eingegangen. Sie werden an dieser Stelle gleichwertig behandelt.

Der Vorschlag der Autoren vom *Simple Indoor Tagging*, Treppen und Fahrstühlen, die von Wänden umgeben sind, das *Tag* „indoor=room“ anstelle von „indoor=area“ zu geben, wurde in Absprache mit dem Fachbetreuer verworfen. Auch die Vorgabe, Fahrstühle mit dem *Tag* „highway=elevator“ zu kennzeichnen, wurde nicht übernommen, weil es sich bei einer Fläche nicht um einen Weg handelt. In der Beschreibung des *Simple Indoor Taggings* wird die Alternative erwähnt, Strukturen, die auf mehreren Etagen vorkommen, nur auf einer Etage zu erfassen. In diesem Fall muss ein zusätzliches *Tag* vergeben werden, das angibt, auf welchen Etagen die Struktur wiederholt auftaucht. Da bereits vor Beginn dieser Arbeit jede Etage des NHGs einzeln dem *Simple-Indoor-Tagging*-Format folgend beschrieben wurde, kann auf diese Alternative ebenfalls nicht eingegangen werden.

```

<way id='1'>
  <nd ref='1' />
  <nd ref='2' />
  <nd ref='3' />
  <nd ref='4' />
  <nd ref='1' />
  <tag k='indoor' v='area' />
  <tag k='stairs' v='yes' />
  <tag k='Level' v='0' />
</way>

<way id='2'>
  <nd ref='5' />
  <nd ref='6' />
  <nd ref='7' />
  <nd ref='8' />
  <nd ref='5' />
  <tag k='indoor' v='area' />
  <tag k='elevator' v='yes' />
  <tag k='Level' v='1' />
</way>

```

Abbildung 6: Treppe (links) und Fahrstuhl im OSM-Format

#### 4.4 Verbindungen zwischen mehreren Etagen

Innerhalb eines Gebäudes befinden sich oft mehrere Treppenhäuser oder Fahrstühle. Im Rahmen dieser Arbeit wurde beschlossen, Relationen anzulegen, in denen jeweils die Treppen oder Fahrstuhlhaltepunkte gespeichert werden, die zusammengehören. Somit kann beim Einlesen der Gebäudedaten ohne zusätzliche Berechnungen festgestellt werden, welche Treppenteile oder Fahrstuhlabschnitte übereinander liegen.

Im Fall von Treppen werden innerhalb der Relation diejenigen Treppen oder Treppenteile als *Members* gespeichert, die miteinander verbunden sind. Dabei werden sie in der Reihenfolge aufgeführt, wie sie miteinander verbunden sind. Es spielt jedoch keine Rolle, ob die Treppe oder das Treppenteil auf der obersten oder untersten Etage als Erstes genannt wird. Handelt es sich um eine Treppe mit einer oder mehreren Verzweigungen, werden nach demselben Muster mehrere Relationen angelegt. Dabei gibt es mehrere Möglichkeiten (vgl. Abb. 7). Wichtig ist lediglich, dass jeder Treppenabschnitt mindestens einmal vorkommt.

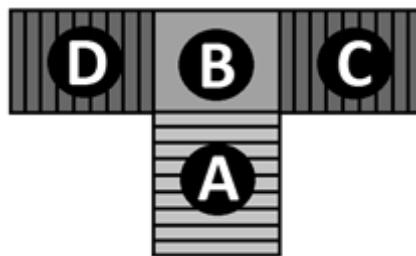


Abbildung 7: Beispiel für eine Treppe mit Verzweigung  
Mögliche Relationen (u.a.): (A, B, C) und (B, D); (D, B, C) und (BA); (A,B), (B,C) und (D,B)

Bei Fahrstühlen erhält die Relation die Polygonzüge der Stelle in jeder Etage, in der der Fahrstuhl halten kann, als *Members*. Die Reihenfolge der *Members* spielt dabei keine Rolle.

Um die Relation als Verbindung zwischen mehreren Etagen zu kennzeichnen, erhält sie Tags mit den Schlüssel-Wert-Paaren „type=connection“ und „connection=stairs“ beziehungsweise „connection=elevator“ (vgl. Abb. 8).

```

<relation id='1'>
  <member type='way' ref='1' />
  <member type='way' ref='2' />
  <member type='way' ref='3' />
  <tag k='connection' ref='elevator' />
  <tag k='type' v='connection' />
</way>

<relation id='2'>
  <member type='way' ref='4' />
  <member type='way' ref='5' />
  <member type='way' ref='6' />
  <tag k='connection' v='stairs' />
  <tag k='type' v='connection' />
</way>
```

Abbildung 8: Verbindungen zwischen mehreren Etagen im OSM-Format  
Oben: Verbindung von mehreren Fahrstuhlebenen; unten: Verbindung zwischen mehreren Treppenabschnitten

## 4.5 Wege innerhalb von Gebäuden

Alle Arten von Wegen und Straßen werden bei *OSM* mit einem Tag mit dem Schlüssel „highway“ als solche gekennzeichnet [OSMI]. Als Wert erhält das Tag eine genauere Beschreibung der Art des Weges. Für Wege, die durch Gebäude führen, gibt es dafür noch keine offizielle Regelung. Es wurde deshalb in dieser Arbeit die Entscheidung getroffen, für Wege, die durch Räume oder Korridore führen, den Wert „footway“ zu vergeben (vgl. Abb. 9). Hiermit werden auch Fußgängerwege im Außenbereich gekennzeichnet [OSMm].

```

<way id='1'>
  <nd ref='1' />
  <nd ref='2' />
  <nd ref='3' />
  <nd ref='4' />
  <nd ref='5' />
  <tag k='highway' v='footway' />
  <tag k='level' v='0' />
</way>
```

Abbildung 9: Weg über eine Etage im OSM-Format

Wege, die über Treppen oder Fahrstühle oder zu diesen hin führen, erhalten stattdessen das Tag „highway=stairs“ beziehungsweise „highway=elevator“. Bei Wegen, die von einer Etage zur nächsten führen, werden beide Etagenwerte durch ein Semikolon getrennt gespeichert. (vgl. Abb. 10)

```
<way id='1'>
  <nd ref='1' />
  <nd ref='2' />
  <nd ref='3' />
  <nd ref='4' />
  <nd ref='1' />
  <tag k='highway' v='stairs' />
  <tag k='Level' v='0;1' />
</way>
```

```
<way id='2'>
  <nd ref='5' />
  <nd ref='6' />
  <nd ref='7' />
  <nd ref='8' />
  <nd ref='5' />
  <tag k='highway' v='elevator' />
  <tag k='Level' v='-1;1' />
</way>
```

Abbildung 10: Wege über mehrere Etagen im OSM-Format

Die Wege in dieser Arbeit werden ausschließlich mithilfe des oben beschriebenen Tools generiert und im entsprechenden Format gespeichert.

## 5 Überarbeitung und Erweiterung des Tools zur Generierung von Wegenetzwerken

Im Rahmen dieser Masterarbeit wurde das Tool zur Wegenetzwerkgenerierung aus dem Forschungsseminar überarbeitet. Zunächst wurde sich für einen objektorientierten Ansatz entschieden, sodass nun die meisten Funktionen den drei Hauptklassen „Room“, „Connection“ und „Parser“ untergeordnet wurden. Des Weiteren wurden einige Funktionen modifiziert und andere hinzugefügt. Auf die Änderungen wird im Folgenden eingegangen.

### 5.1 Hilfsfunktionen

In diesem Abschnitt werden einige Hilfsfunktionen vorgestellt, die klassenübergreifend verwendet werden. Es wird sich dabei auf Funktionen beschränkt, die während des Forschungsseminars noch nicht verwendet wurden.

#### 5.1.1 Überprüfen, ob sich ein Punkt innerhalb des Raumes befindet

Mithilfe der Funktion *point\_inside\_room* wird untersucht, ob sich ein einzelner Punkt innerhalb des Raumes befindet. Das bedeutet, dass sich der Punkt innerhalb des Polygons, das den Raum darstellt, befinden muss. Der Punkt wird auch als außerhalb des Polygons liegend definiert, wenn er sich auf einer der Kanten oder Eckpunkte desselben befindet. Befinden sich innerhalb des Raumes Barrieren, darf der Punkt außerdem nicht innerhalb eines Polygons, das eine Barriere darstellt, oder auf einer der Kanten oder Eckpunkte dieses Polygons liegen.

Der Algorithmus, der testet, ob sich der Punkt innerhalb eines Polygons befindet, orientiert sich an der bei Sutherland et al. [SSS74, S.13] beschriebenen Strahlmethode. Dabei wird ausgehend von dem Punkt ein Strahl in eine beliebige Richtung gezogen und die Anzahl der Schnittpunkte mit dem Polygon gezählt (ebd.). Gibt es eine ungerade Anzahl an Schnittpunkten zwischen dem Strahl und dem Polygon, befindet sich der Punkt innerhalb des Polygons (ebd.; vgl. Abb. 11). Die Methode funktioniert nicht, falls sich der Punkt auf einer der Kanten des Polygons befindet (ebd.). Ein weiteres Problem tritt auf,

wenn der Strahl einen der Eckpunkte des Polygons schneidet ([HA01, S. 137]; vgl. Abb. 12).

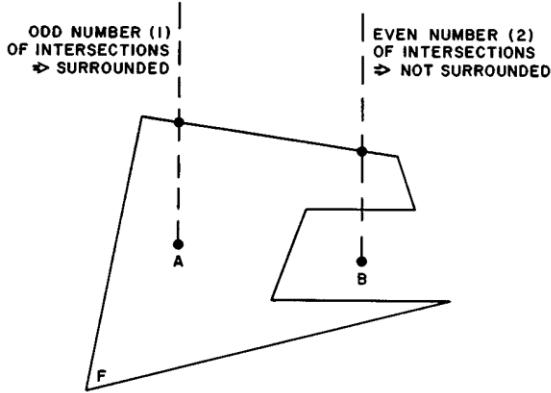


Abbildung 11: Verdeutlichung der Strahlmethode  
(Quelle: Sutherland et al [SSS74, S.14])

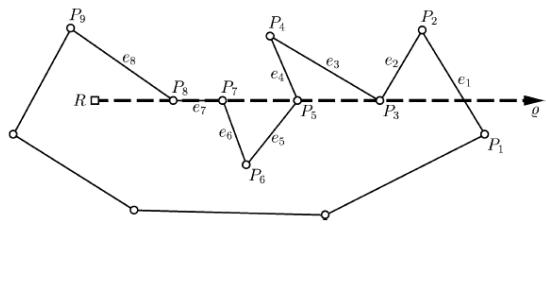


Abbildung 12: Probleme bei der Strahlmethode  
(Quelle: Hormann & Agathos [HA01, S. 137])

In Abbildung 12 würde der Algorithmus ein falsches Ergebnis zurückgeben, falls einer der Punkte, die der Strahl schneidet, entfernt werden würde. Sunday [Sun] wandelte den Algorithmus so ab, dass die Orientierung der Kanten berücksichtigt wird. Der Strahl verläuft von dem zu untersuchenden Punkt ausgehend parallel zur x-Achse nach rechts. Die Kanten des Polygons werden einzeln nacheinander betrachtet. Handelt es sich um eine aufwärtsverlaufende Kante, wird ein eventueller Schnittpunkt mit dem Endpunkt der Kante nicht mitgezählt. Bei abwärtsverlaufenden Kanten werden dagegen Schnittpunkte mit dem Anfangspunkt der jeweiligen Kante ignoriert. Kanten, die parallel zur x-Achse verlaufen, werden überhaupt nicht untersucht. Dieser Algorithmus von Sunday wurde für diese Arbeit nachprogrammiert. Der einzige Unterschied besteht darin, dass ein Punkt immer als außerhalb des Polygons definiert wird, wenn er sich auf einer Kante desselben befindet.

### 5.1.2 Überprüfen, ob sich ein Weg innerhalb des Raumes befindet

Die Funktion `way_inside_room` wurde geschrieben, um herauszufinden, ob sich ein Weg vollständig innerhalb eines Raumes befindet. In ihr wird der Weg auf zwei Kriterien untersucht (vgl. Abb. 13):

1. Die Streckenmittelpunkte der Teilabschnitte des Weges befinden sich innerhalb des Raumes.
2. Der Weg schneidet an keiner Stelle eine Wand oder eine Barriere innerhalb des Raumes.

Nur wenn beide Kriterien erfüllt sind, kann sichergestellt werden, dass sich der Weg innerhalb des Raumes befindet.

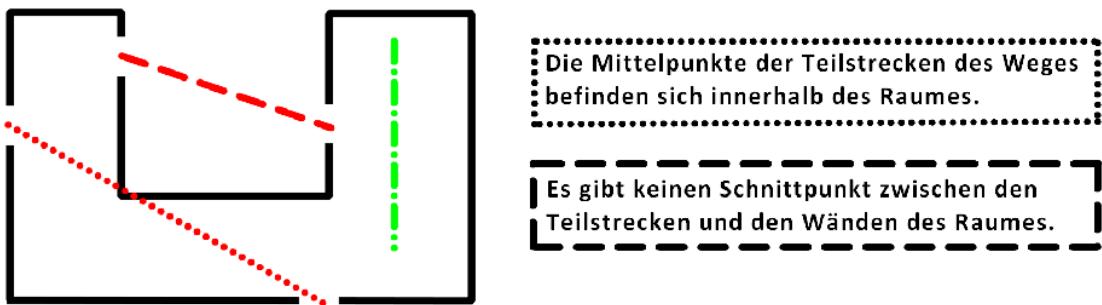


Abbildung 13: Verdeutlichung der Kriterien zur Bestimmung, ob sich ein Weg innerhalb eines Raumes befindet

## 5.2 Die Klasse *Room*

Mithilfe der Klasse *Room* werden Räume und Korridore<sup>6</sup> gespeichert und mögliche Wege innerhalb dieser berechnet. Eine Instanz der Klasse *Room* wird mit Hilfe des Polygonzugs eines Raums und dem Wert der Etage, auf dem dieser sich befindet, erstellt. Falls sich innerhalb des Raumes Barrieren, etwa Säulen oder Trennwände, befinden, werden die Polygonzüge dieser ebenfalls übergeben. Alle Werte werden als Attribute der Klasse gespeichert. Sie werden für die Berechnung von Wegen innerhalb der Räume benötigt.

### 5.2.1 Wege generieren auf Basis des *Straight Skeletons*

In der Funktion `find_ways` werden Wege mit Hilfe des *Straight Skeletons*<sup>7</sup> generiert. Dabei werden nur die Verbindungen des *Straight Skeletons*, die zu Türen führen oder nicht

---

<sup>6</sup> Da innerhalb des Programms zwischen Korridoren und Räumen nicht unterschieden wird, wird nachfolgend nur von Räumen gesprochen.

<sup>7</sup> Ein *Straight Skeleton* kann man sich wie ein Dach vorstellen, das die Fläche eines Polygons überspannt [HS08, S. 172]. Genauereres kann in der Forschungsseminararbeit [Aus18] nachgelesen werden.

aus Punkten des Polygons bestehen, als Wege gespeichert (vgl. [Aus18, S.7f.]). Berechnet werden die Wege mit der Funktion *skeletonize* der Bibliothek *polyskel*. Der Funktion wurde bisher der Polygonzug eines Raumes übergeben. Im Forschungsseminar wurde im Ausblick erwähnt, dass in Zukunft auch auf unüberwindbare Elemente innerhalb eines Raums eingegangen werden sollte, da sonst ungültige Wege entstehen können (vgl. Abb. 14, links). Da der Funktion auch Löcher im Polygon übergeben werden können (vgl. Abb. 14, rechts), wurde davon ausgegangen, dass dies kein Problem darstellen sollte.

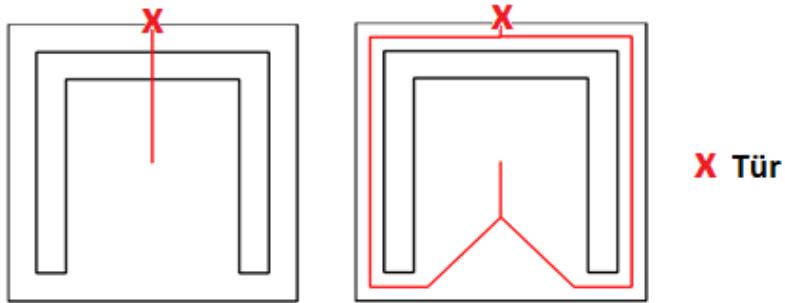


Abbildung 14: Vergleich zwischen Straight Skeletons mit und ohne Übergabe der Barriere als Loch

Um Wege zu berechnen, werden nun der Funktion *skeletonize* der gespeicherte Polygonzug des Raums und, sofern vorhanden, die Polygonzüge der Barrieren übergeben. Anschließend werden nur die Verbindungen, die zu Türen oder zu Punkten innerhalb des Raumes und außerhalb der Barrieren führen, als Wege gespeichert.

### 5.2.2 Wege vereinfachen

Die Funktion *simplify\_ways* wurde geschrieben, um generierte Wege zu vereinfachen. Bisher entschied die Wahl eines Schwellenwerts über die Qualität der Ergebnisse. Es zeigte sich jedoch, dass es keinen optimalen Schwellenwert gibt, sodass der Schwellenwert für jeden Weg einzeln gewählt werden müsste (vgl. [Aus18, S.10f.]).

Die Funktion wurde so angepasst, dass nun auf die Wahl eines Schwellenwerts verzichtet werden kann. Wie im ursprünglichen Algorithmus werden immer drei aufeinander folgende Knoten eines Weges untersucht. Der erste und dritte Knoten werden als potentieller Weg gemeinsam mit dem Polygon des Raums und gegebenenfalls den Barrieren an die

oben beschriebene Hilfsfunktion `way_inside_room` übergeben. Befindet sich der potentielle Weg innerhalb des Raumes, wird der zweite Knoten entfernt (vgl. Abb. 15). Dieser Vorgang wird wiederholt, bis der Weg nicht weiter vereinfacht werden kann.

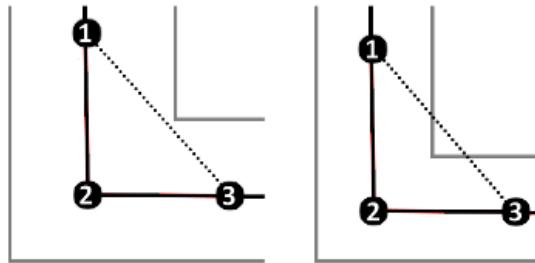


Abbildung 15: Funktionsweise von `simplify_ways`  
Links würde der Knoten 2 entfernt werden, rechts nicht

### 5.2.3 Direkte Verbindungen zwischen Türen suchen

Im Forschungsseminar wurde festgestellt, dass es bei der automatischen Generierung von Navigationsgraphen zwei grundsätzliche Herangehensweisen gibt (vgl. Abb. 16). Bei der ersten werden Wege generiert, die annähernd auf Mittellinien verlaufen. Die andere Herangehensweise bevorzugt Wege, die bestenfalls direkt von Tür zu Tür verlaufen. Dieser zweite Ansatz wird von Liu und Zlatanova [LZ11] als *Door-to-Door*-Ansatz bezeichnet. Falls eine direkte Verbindung zwischen zwei Türen nicht vorhanden ist, wird ein Weg über die konkaven Ecken des Raumes gesucht [LZ11, S. 3].

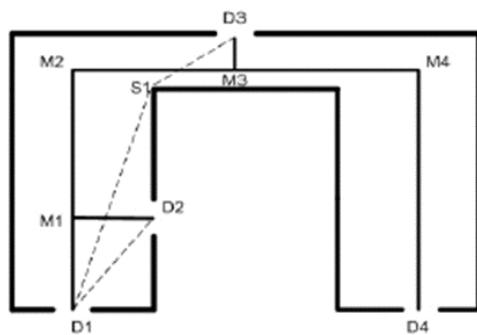


Abbildung 16: Vergleich zwischen Wegen, die von Tür zu Tür bzw. auf Mittellinien verlaufen  
(Quelle: Liu & Zlatanova, [LZ11, S. 2])

Im Forschungsseminar war es aus Zeitgründen nicht möglich, beide Ansätze zu implementieren. Es wurde sich für den Ansatz der Wege über Mittellinien entschieden, weil der dabei gebildete Navigationsgraph als übersichtlicher erschien. Es wurde aber für sinnvoll gehalten, zukünftig zusätzlich auch den *Door-to-Door*-Ansatz bei der Generierung von Navigationsgraphen mit einzuschließen.

Im Algorithmus von Liu und Zlatanova wird als Erstes berechnet, welche Räume in welcher Reihenfolge durchquert werden müssen, um vom Ausgangspunkt das Ziel zu erreichen [LZ11, S. 3]. Ausgehend von dem Raum, in dem sich der Ausgangspunkt befindet, wird die Route Schritt für Schritt aufgebaut. Es wird untersucht, ob es im aktuellen Raum eine Tür in Sichtweite gibt, die in den nächsten zu durchquerenden Raum führt. Gibt es mehrere Türen, auf die das zutrifft, wird die nächstgelegene gewählt. Gibt es keine passende Tür in Sichtweite, wird der kürzeste Weg in den nächsten Raum über die konkaven Ecken des Raumes berechnet. Diese Schritte werden wiederholt, bis der Zielraum erreicht wurde. Dieser Algorithmus kann dazu führen, dass verschiedene Routen entstehen, wenn Ausgangs- und Zielpunkt vertauscht werden ([LZ11, S. 5]; vgl. Abb. 17).

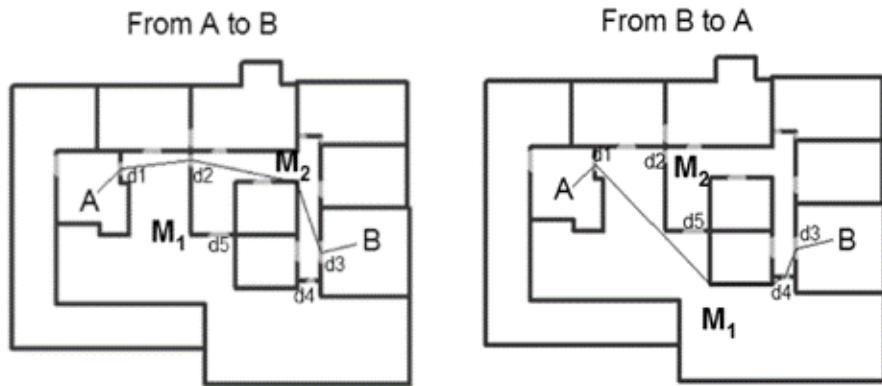


Abbildung 17: Verschiedene Routen bei Vertauschung von Ausgangs- und Zielpunkt beim Door-to-Door-Algorithmus nach Liu und Zlatanova  
(Quelle: Liu & Zlatanova, [LZ11, S. 5])

Liu und Zlatanova bevorzugen eine Generierung von Routen, die der „„natürlichen“ Bewegung von Fußgängern“ [LZ11, S. 1] entspricht. Das Ziel dieser Arbeit ist jedoch, dem Benutzer die Navigation in Gebäuden zu erleichtern. Es spricht nichts dagegen, dem Benutzer eine Route vorzuschlagen, die er normalerweise nicht eingeschlagen hätte. Im Gegenteil wird dies in dieser Arbeit sogar als Vorteil gewertet, weil das Indoor-Routing den Benutzer dadurch unterstützt, sein Ziel schneller zu erreichen.

Der Fokus liegt auf der übersichtlichen Darstellung der Route zum Erreichen des Ziels. Aus diesem Grund wurde auch entschieden, falls möglich keine Wege zu generieren, die direkt an den Wänden verlaufen. Stattdessen wurde der Kompromiss eingegangen, auf die konkaven Ecken des Raumes zu verzichten und nur nach direkten Verbindungen zwischen Türen zu suchen. Für diesen Zweck wurde die Funktion *door\_to\_door* geschrieben.

Die Funktion wird nur ausgeführt, wenn mehr als eine Tür in den Raum führt. In diesem Fall werden alle Türen des Raumes paarweise in allen möglichen Kombinationen betrachtet. Falls die Verbindung zwischen zwei Türen sich innerhalb des Raumes befindet und keine Barriere durchkreuzt, wird sie als neuer Weg des Raumes gespeichert.

#### 5.2.4 Sich kreuzende Wege teilen

Bei der Generierung von Wegen, vor allem bei Verwendung des *Door-to-Door*-Algorithmus, kommt es vor, dass die berechneten Wege sich überschneiden (vgl. Abb. 18). Bei einem Graphen ist es nur möglich, von einer Kante auf die andere zu wechseln, wenn die Kanten sich einen Knoten teilen. Genauso kann unter OSM nur von einem Weg auf den anderen gewechselt werden, wenn beide sich in der OSM-Notation einen *Node* teilen. Aus diesem Grund wurde die Funktion *split\_ways* geschrieben. Sie überprüft alle Wege darauf, ob sie sich mit einem anderen Weg überkreuzen. Gegebenenfalls werden die zwei kreuzenden Wege in vier Wege aufgeteilt (vgl. Abb. 18).

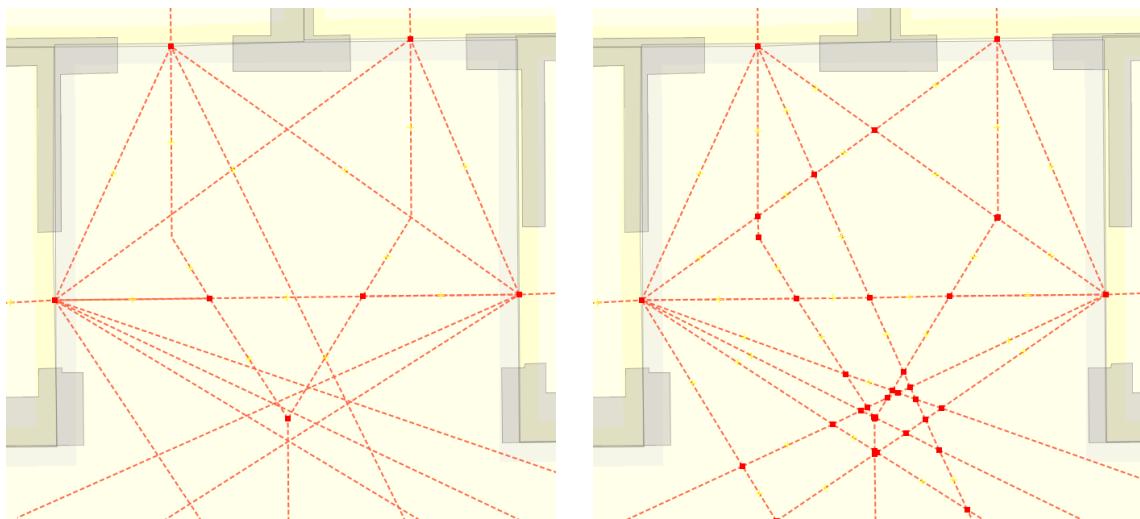


Abbildung 18: sich kreuzende Wege (Ausschnitt aus generierten Wegen für das NHG)  
Markiert wurden jeweils der erste und letzte Knoten eines Weges.  
Links: ohne Aufruf von *split\_ways*; rechts: nach Aufruf von *split\_ways*

### 5.3 Die Klasse *Connection*

In der Klasse *Connection* werden die Verbindungen zwischen mehreren Etagen als Wege gespeichert. Um eine Instanz der Klasse zu erstellen, müssen die Art der Verbindung (Treppe oder Fahrstuhl) und eine Liste der einzelnen Verbindungsabschnitte, aus der

diese sich zusammensetzt, übergeben werden. Jeder einzelne Verbindungsabschnitt besteht aus einer Liste von Punkten, die die Fläche der Treppe oder des Fahrstuhls darstellen, und dem Wert der Etage, auf dem diese sich befindet.

### 5.3.1 Wege über Treppen und Fahrstühle speichern

Die Funktion *add\_connections* wurde geschrieben, um die Verbindungen zwischen zwei Etagen und die Verbindungen einer Treppe beziehungsweise eines Fahrstuhls<sup>8</sup> zu einem Raum als Weg zu speichern.

Für jedes Polygon, das einen Verbindungsabschnitt abbildet, wird zunächst als repräsentativer Punkt der Schwerpunkt gebildet. Bei Treppen handelt es sich hierbei um eine provisorische Lösung, da diese zu diesem Zeitpunkt noch nicht genauer kartographiert werden.

Wie oben beschrieben, werden Verbindungselemente mithilfe einer Tür mit dem angrenzenden Raum verbunden. In nächsten Schritt werden deshalb für das aktuell betrachtete Verbindungselement alle angrenzenden Türen bestimmt. Dies funktioniert auf dieselbe Art wie die Suche nach Eingängen, die in Räume oder Korridore führen (vgl. [Aus18, S. 7]).

Für jede Tür, die zu einem Verbindungselement führt, wird ein neuer Weg angelegt, der aus den Koordinaten der Tür und des Schwerpunkts des jeweiligen Verbindungselements besteht. Ebenso werden der Wert der Etage und die Art des Verbindungselements gespeichert.

Bei der Verbindung zwischen zwei Etagen unterscheidet der Algorithmus, ob es sich bei dem Verbindungselement um mehrere Treppen oder um einen Fahrstuhl handelt. Handelt es sich um eine Verbindung zwischen Treppen, werden die Koordinaten der Schwerpunkte von jeweils zwei aufeinander folgenden Treppen als Weg gespeichert. Handelt es sich um einen Fahrstuhl, werden die Koordinaten der Schwerpunkte aller Fahrstuhlhalbschnitte paarweise untereinander kombiniert und als Wege gespeichert. In beiden Fällen

---

<sup>8</sup> Im Folgenden werden Treppen und Fahrstühle verallgemeinernd als Verbindungselemente bezeichnet.

len werden zusätzlich die Etagenwerte beider Schwerpunkte durch ein Semikolon getrennt in einem Wert gespeichert. Dabei muss der als Etagenwerte des Knoten zuerst genannt werden, der auch als Erstes gespeichert wird.

## 5.4 Die Klasse *Parser*

Die Klasse *Parser* ist die übergeordnete Klasse. In ihr befinden sich die Funktionen zum Einlesen der Gebäudedaten und zum Schreiben der gefundenen Wege ins *OSM*-Format. Der *Parser* berücksichtigt beim Einlesen der Daten neben *Nodes* und *Ways*, die Eingänge, Räume oder Korridore darstellen, nun auch Multipolygon-Relationen und Relationen, die Verbindungen zwischen den Etagen abbilden. Je nachdem, um was für ein Element es sich handelt, wird ein Eingang gespeichert, eine Instanz der Klasse *Room* oder eine Instanz der Klasse *Connection* erstellt.

Die Räume, die beim Parsen von Multipolygon-Relationen gefunden wurden, wurden ebenfalls beim Parsen der *Ways* gefunden. Für diese Räume wurden also zwei Instanzen der Klasse *Room* erstellt. Nachdem das gesamte Dokument geparsst wurde, muss deswegen nach doppelt gespeicherten Räumen in der Liste der gefundenen Räume gesucht werden. Es wird jeweils die Instanz der Klasse, die ohne Barrieren angelegt wurde, wieder aus der Liste entfernt.

## 5.5 Funktionsweise des Tools nach Übernahme der Änderungen

Bei Ausführung des Programms müssen die Namen der *OSM*-Datei, die die Gebäudedaten enthält, und der Datei, in der die generierten Wege gespeichert werden sollen, als Kommandozeilenargumente übergeben werden. Optional können ebenfalls „sw“ und „dd“ als Kommandozeilenargumente übergeben werden. Sie stehen für „*simplify\_ways*“ und „*door\_to\_door*“. Bei Vorhandensein werden die entsprechenden Funktionen ausgeführt.

Die jetzige Funktionsweise des Tools zur automatischen Generierung von Wegenetzwerken kann im untenstehenden Flussdiagramm nachvollzogen werden.

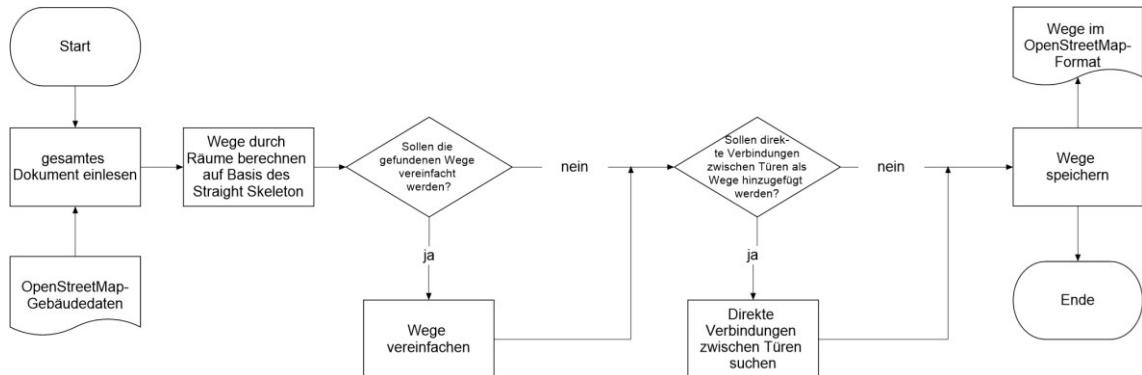


Abbildung 19: Funktionsweise des Tools zur automatischen Generierung von Wegenetzwerken

## 5.6 Auswertung der generierten Wege

In diesem Abschnitt werden die Wege, die vom Parser generiert werden, in einem ersten Schritt ausgewertet. Es wird getrennt auf die Probleme eingegangen, die bei der Generierung der Wege mit Hilfe des *Straight Skeletons* und durch den *Door-to-Door*-Algorithmus entstanden.

### 5.6.1 Durch *Straight Skeleton* generierte Wege

Bei der Anwendung der Funktion zur automatischen Generierung von Wegen auf Basis des *Straight Skeletons* auf den Gebäudeplan des NHGs zeigte sich, dass bei vorliegenden Barrieren die dabei entstehenden Wege nicht optimal sind. Zwar ist gewährleistet, dass die entstandenen Wege nicht durch die Barrieren führen, es fehlen dafür aber wichtige Verbindungen. So werden Wege berechnet, bei denen manche Eingänge nicht erreichbar sind (vgl. Abb. 20). Des Weiteren entstehen isolierte Wege, die mit keinen anderen Wegen verbunden sind (vgl. Abb. 20, rechts). In anderen Fällen ist erst bei einer starken Vergrößerung sichtbar, dass es sich nicht um einen Weg, sondern zwei sehr nah beieinander verlaufende Wege handelt (vgl. Abb. 21). Sollte sich der Benutzer bei der Ortung innerhalb des Gebäudes in der Nähe solcher Wege befinden, könnte dies zu Umwegen in der Routenplanung führen. Keines dieser Probleme trat bei Räumen oder Korridoren ohne Barrieren auf.

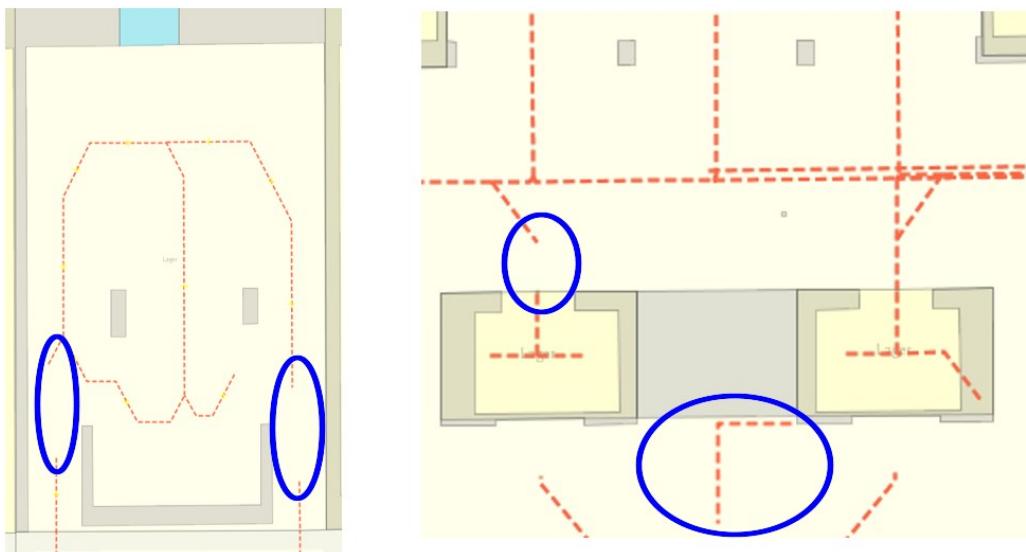


Abbildung 20: Wege, die nicht mit Eingängen verbunden sind, und ein isolierter Weg

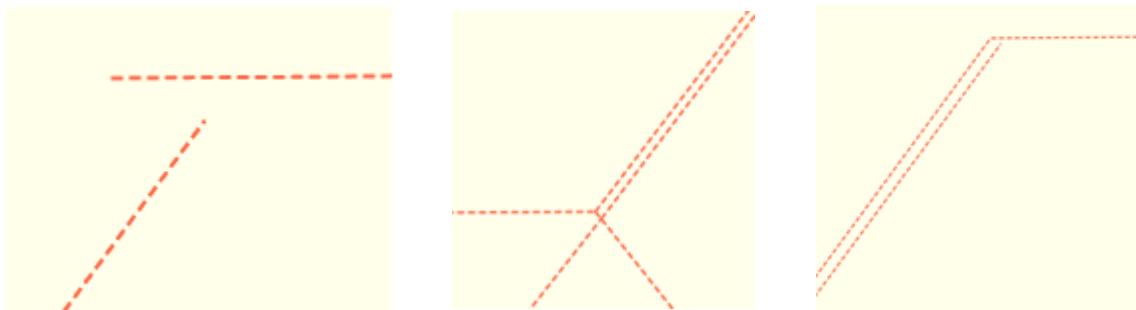


Abbildung 21: Dicht beieinander verlaufende Wege, die zu Umwegen führen können

Bei genauerer Betrachtung zeigte sich, dass Barrieren, die dicht beieinander liegen oder sich überlappen, die Qualität der generierten Wege stark beeinflussen können. In Abbildung 22 links sind ein Polygon mit drei direkt nebeneinander liegenden Löchern und das durch *polyskel* generierte *Straight Skeleton* zu sehen. Die Verbindungen des *Straight Skeletons* zu Eckpunkten des Polygons oder der Löcher wurden der Übersichtlichkeit halber weggelassen. Das Ergebnis ist für die Generierung von Wegen unbrauchbar. In Abbildung 22 rechts ist das *Straight Skeleton* zu sehen, das generiert wird, wenn statt der einzelnen drei Löcher der Umriss der drei Löcher übergeben wird. Das dabei entstehende *Straight Skeleton* kann für die Generierung von Wegen verwendet werden.

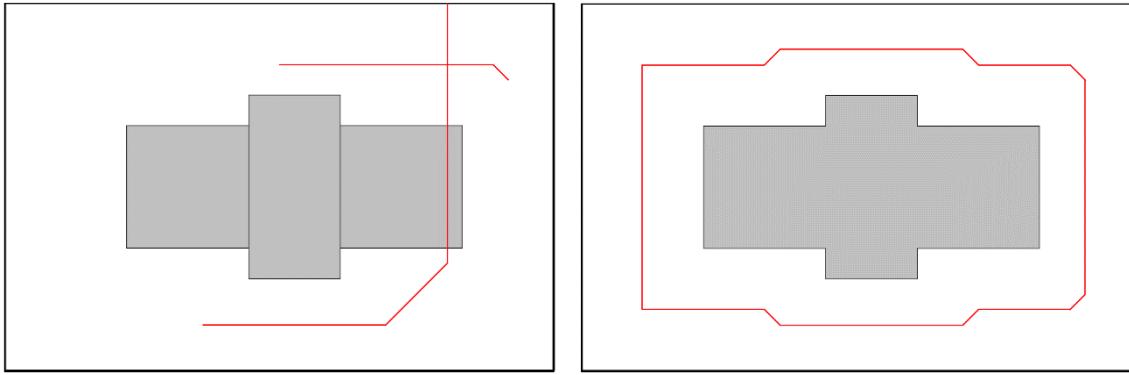


Abbildung 22: Straight Skeleton bei der Übergabe von Löchern  
Links: drei einzelne Löcher; rechts: drei Löcher zu einem Loch zusammengefasst

Auch für die Barrieren im NHG wurden probehalber direkt nebeneinander liegende Barrieren zu einer Barriere zusammengefasst. Dies hatte zur Folge, dass die fehlende Verbindung in Abbildung 20 rechts nicht mehr vorhanden war. Außerdem wurden keine sehr dicht nebeneinander verlaufenden Wege wie in Abbildung 21 gebildet. Das Problem der fehlenden Verbindungen zu Eingängen wie in Abbildung 20 links konnte dadurch jedoch nicht gelöst werden. Aus diesem Grund ist es unverzichtbar, die automatisch generierten Wege noch einmal manuell zu überprüfen. Es ist dennoch ratsam, in Zukunft eine Funktion zu entwickeln, die nahe beieinander liegende Barrieren zu einer Barriere zusammenfasst. Alternativ könnte eine bessere *Straight-Skeleton*-Implementierung verwendet werden.

Um isolierte Wege aus den zu speichernden Wegen zu entfernen, wurde eine entsprechende Funktion geschrieben. Als Übergangslösung für fehlende Verbindungen wurde die Funktion *add\_supplementary\_ways* geschrieben, die zunächst für jeden Weg innerhalb eines Raumes den ersten und den letzten Knoten bestimmt. Für alle gefundenen Knoten wird anschließend paarweise untersucht, ob sie einen gültigen Weg darstellen. Dabei werden nur potentielle Verbindungen untersucht, bei denen die zwei Knoten nicht aus demselben Ursprungsweg stammen. Um das Netzwerk nicht zu dicht zu gestalten, werden des Weiteren nur Verbindungen als neue Wege gespeichert, die keinen bestehenden Weg kreuzen. Wie in Abbildung 23 zu sehen ist, entstehen dabei Wege, die dicht an den Wänden entlang führen. Aus diesem Grund sollte in Zukunft eine andere Lösung gefunden werden.

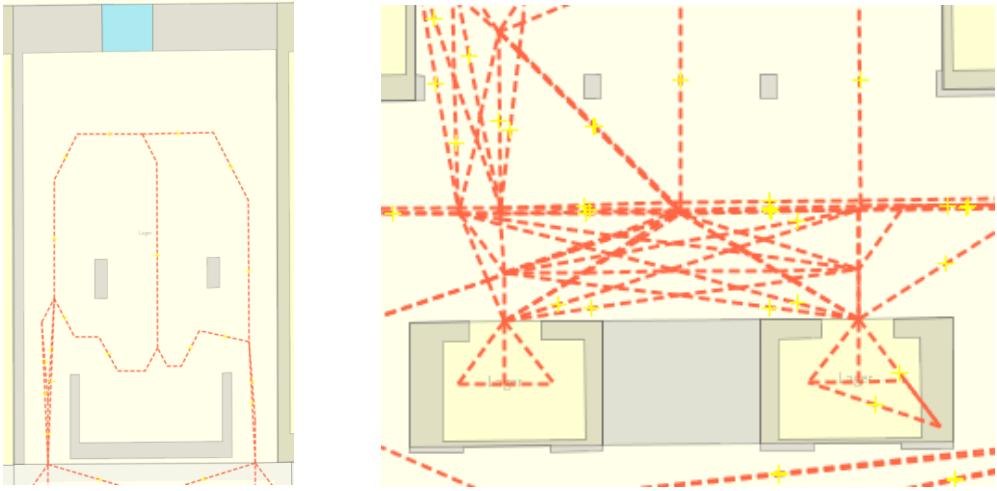


Abbildung 23: Generierte Wege nach dem Aufruf der Funktion `add_supplementary_ways` (vgl. Abb. 21)

### 5.6.2 Durch *Door-to-Door*-Algorithmus generierte Wege

Beim *Door-to-Door*-Algorithmus werden nur gültige Wege gebildet. Es entsteht jedoch ein deutlich dichteres Netzwerk als bei der alleinigen Anwendung des *Straight-Skeleton*-Algorithmus (vgl. Abb. 24). Des Weiteren gibt es Wege, die sehr dicht an den Wänden entlang führen. Wie oben erwähnt, werden solche Wege als weniger übersichtlich erachtet. Beim Einsatz des Routenplaners sollte deshalb auch geprüft werden, ob bei einem Wegenetzwerk, das durch die Wege des *Door-to-Door*-Algorithmus ergänzt wurde, zufriedenstellendere Routen berechnet werden als bei einem Wegenetzwerk ohne diese zusätzlichen Wege.

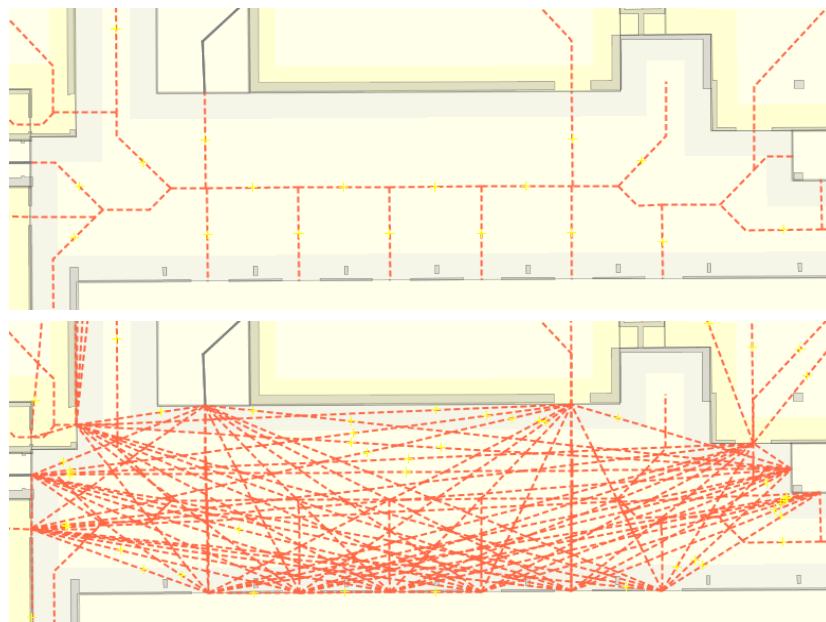


Abbildung 24: Generierte Wege mit und ohne *Door-to-Door*-Algorithmus  
(Die Funktion `add_supplementary_ways` wurde nicht aufgerufen.)

## 6 Auswahl und Anpassung einer Routingmaschine für den Indoorbereich

Bei der Auswahl einer Routingsoftware, die für die Anwendung innerhalb von Gebäuden angepasst werden kann, wurden mehrere Kriterien beachtet. Die erste Bedingung war, dass es sich um eine quelloffene Software handeln muss. Zudem muss die Routingsoftware auf Basis von *OSM*-Daten funktionieren. Darüber hinaus sollte es möglich sein, verschiedene Routingprofile zu erstellen, um auf die individuellen Bedürfnisse der Nutzer eingehen zu können. Das Framework, in das die Routingmaschine später integriert werden soll, ist in Java geschrieben. Deshalb bot es sich an, eine Routingsoftware in derselben Sprache auszuwählen. Da es auf lange Sicht sinnvoll erscheint, ein hybrides Routing zwischen Indoor- und Outdoorbereich wie bei [WFH+17] zu ermöglichen, sollten möglichst viele Fortbewegungsmittel unterstützt werden.

Im offiziellen *OSM*-Wiki sind mehrere quelloffene Routingmaschinen nach Programmiersprachen geordnet aufgelistet [OSMk]). Darunter fallen vier in Java geschriebene Routingmaschinen (ebd.). Die Wahl fiel auf *GraphHopper*, weil die Routingsoftware die einfache Erstellung von verschiedenen Routingprofilen ermöglicht und bereits viele Fortbewegungsmittel unterstützt. Sie hat sich außerdem schon insofern bewährt, als dass sie neben der in C++ geschriebenen Software *Open Source Routing Machine* beim Routing auf der offiziellen Seite von *OpenStreetMap* verwendet wird. Ein weiterer Vorteil ist, dass sie vielseitig einsetzbar ist, weil sie bereits jeweils für Onlineanwendungen, mobile Anwendungen und Desktopapplikationen optimiert wurde.

Die folgenden Abschnitte widmen sich der grundsätzlichen Funktionsweise des Routings unter *GraphHopper*<sup>9</sup>. Dabei wird auf die einzelnen Schritte nacheinander eingegangen. Als Erstes wird jeweils der aktuelle Schritt überblicksartig vorgestellt. Anschließend wird erklärt, welche bestehenden Klassen erweitert oder hinzugefügt werden mussten, um den Schritt dem Indoor-Routing anzupassen.

---

<sup>9</sup> Diese Arbeit stützt sich auf den [Commit ebf2634](#) von *GraphHopper* vom 07.05.2018.

## 6.1 Erstellen eines Navigationsgraphen

Bei der ersten Nutzung von *GraphHopper* für das Routing in einem bestimmten Bereich werden die Daten dieser Region als *OSM*-Datei übergeben. Beim Einlesen der Datei wird aus den gefundenen Wegen und Straßen ein Navigationsgraph erstellt.

*GraphHopper* unterscheidet beim Einlesen von *Ways*, die Wege darstellen, zwischen *Pillar Nodes* und *Tower Nodes*. Der erste und letzte *Node* eines gefundenen *Ways* werden als *Tower Nodes* des Graphen gespeichert (vgl. Abb. 25). Als *Pillar Nodes* werden die *Nodes* bezeichnet, die zwischen den *Tower Nodes* liegen.

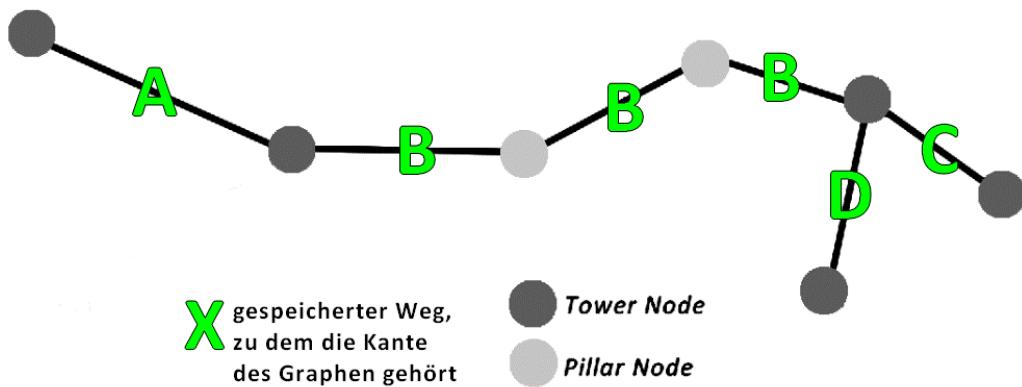


Abbildung 25: Pillar Nodes und Tower Nodes

Für jeden Knoten des Graphen wird die genaue geographische Position gespeichert, so dass jede Kantenlänge der tatsächlichen Entfernung zwischen den zwei Knoten entspricht, die die Kante verbindet. Um eine schnellere Berechnung des kostengünstigsten Pfads zwischen zwei Punkten zu ermöglichen, werden Kanten zusammengefasst. Befinden sich zwischen zwei *Tower Nodes* ein oder mehrere *Pillar Nodes*, wird eine Kante gespeichert, die lediglich aus den beiden *Tower Nodes* besteht. Als Länge der Kante wird die Summe der Längen aller Kanten, die zwischen den beiden *Tower Nodes* liegen, gespeichert. Die Informationen dazu, welche *Pillar Nodes* zwischen welchen *Tower Nodes* liegen, werden gesondert gespeichert.

Für jedes ausgewählte Fortbewegungsmittel wird einzeln untersucht, welche Kanten bei der Routenplanung berücksichtigt werden dürfen und mit welcher durchschnittlichen Geschwindigkeit diese vermutlich überquert werden. Der gesamte Navigationsgraph wird codiert lokal gespeichert, sodass die Daten bei der nächsten Benutzung nicht erneut eingelesen werden müssen.

### 6.1.1 Speichern der Etagenwerte

Bei einem Routing innerhalb von Gebäuden über mehrere Etagen hinweg muss für den Benutzer ersichtlich sein, auf welchen Etagen sich die einzelnen Routenabschnitte befinden. Standardmäßig werden für jeden Knoten innerhalb des Navigationsgraphen ein Längen- und ein Breitengrad codiert gespeichert. Optional kann auch die Höhe des Knoten gespeichert werden<sup>10</sup>. Um einen weiteren Wert für einen Knoten oder für eine Kante speichern zu können, bietet *GraphHopper* die Möglichkeit, eine sogenannte Grapherweiterung zu verwenden. In dieser Arbeit wurde eine Grapherweiterung implementiert, die für jeden *Tower Node* im Graphen zusätzlich den Wert der Etage speichert, auf dem dieser sich befindet. In dem vom Tool generierten Navigationsgraph kommen *Pillar Nodes* nur zwischen *Tower Nodes* vor, die sich auf derselben Etage befinden. Befinden sich zwei *Tower Nodes* auf unterschiedlichen Etagen, befinden sich keine *Pillar Nodes* zwischen ihnen. Die Etagenwerte der *Pillar Nodes* lassen sich also immer aus den *Tower Nodes* ableiten und müssen deshalb nicht zusätzlich gespeichert werden.

### 6.1.2 Hinzufügen von *Flag Encodern*

Um die gültigen Kanten des Navigationsgraphen für ein Fortbewegungsmittel zu ermitteln, müssen sogenannte *Flag Encoder* implementiert worden sein. In einem *Flag Encoder* wird festgelegt, welche *Tags* einen Weg im OSM-Format als relevant oder irrelevant für ein Fortbewegungsmittel einstufen. Zusätzlich speichert der *Flag Encoder* für jede gefundene gültige Kante die durchschnittliche Geschwindigkeit, mit der sich der Benutzer auf dieser Kante des Graphen unter Berücksichtigung von erlaubten Höchstgeschwindigkeiten fortbewegen wird.

Im Hinblick auf das Ziel, ein Indoor-Routing für körperlich Beeinträchtigte zu schaffen, wurden in dieser Arbeit drei *Flag Encoder* hinzugefügt. Der erste berücksichtigt alle Wege innerhalb des Gebäudes. Der zweite schließt die Wege, die über Treppen führen, aus, während der letzte solche ausschließt, die über Fahrstühle führen. Als durchschnittliche Geschwindigkeit wurde die aus dem *Flag Encoder* für Fußgänger übernommen. Die Routenlänge wird jedoch dadurch verfälscht, dass die Verbindungen zwischen zwei Eta-

---

<sup>10</sup> Dies ist sinnvoll, wenn die Steigung von Wegen eine Rolle spielt, zum Beispiel bei einem Routing für Wanderer.

gen im Navigationsgraph eine Kantenlänge von 0 Metern haben können, wenn die jeweiligen Knoten direkt übereinander liegen. Hinzu kommt, dass das Tempo von Personen, die in Gebäuden unterwegs sind, sehr unterschiedlich sein kann. Deshalb sollte die Gesamtdauer, die für eine Route errechnet wird, bis auf Weiteres ignoriert werden.

## 6.2 Suchanfrage starten

Bei Vorliegen eines Navigationsgraphen kann der Benutzer die Koordinaten eines Startpunkts und eines Zielpunkts angeben. Bei Bedarf kann er außerdem beliebig viele Zwischenziele hinzufügen. Sobald der Benutzer eine Suchanfrage abschickt, wird für jeden der übergebenen Punkte die am nächsten liegende Kante des Graphen bestimmt. Berücksichtigt werden dabei auch die Kanten, die aus *Pillar Nodes* bestehen. Die Stelle auf der gefundenen Kante, die sich am nächsten an dem betrachteten Punkt befindet, wird als angenäherter Punkt gespeichert (vgl. Abb. 26)

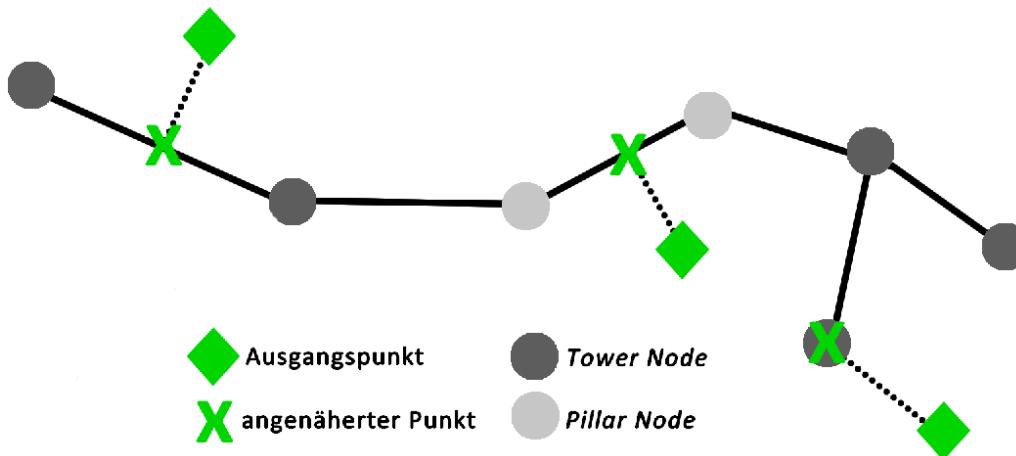


Abbildung 26: Annäherungen von übergebenen Punkten bei der Suchanfrage

### 6.2.1 Kanten des Graphen filtern

Bei der Suche nach der nächstliegenden Kante zu einem übergebenen Punkt können die Kanten des Graphen, die betrachtet werden sollen, bei Bedarf mithilfe eines Kantenfilters gefiltert werden. In dieser Arbeit wurde ein Kantenfilter implementiert, der nur die Kanten zulässt, die sich auf derselben Etage wie der übergebene Punkt befinden. Für jeden Punkt in der Suchanfrage muss also auch die Etage übergeben werden, auf der dieser sich befindet.

## 6.3 Route berechnen

Ausgehend von den angenäherten Punkten wird mithilfe eines voreingestellten Routingalgorithmus der Weg mit den geringsten Kosten berechnet. Werden mehr als zwei Punkte übergeben, wird für jeweils zwei aufeinanderfolgende Punkte die kostengünstigste Route ermittelt. Derzeit wird keine Kantengewichtung verwendet, sodass es sich bei dem Weg mit den geringsten Kosten um den kürzesten handelt.

Jeder Routingalgorithmus liefert als Ergebnis für die Route zwischen zwei Punkten eine Instanz der Klasse *Path* zurück. Mithilfe der Methode *calcPoints* innerhalb dieser Klasse wird eine Instanz der Klasse *PointList* erstellt. Sie enthält die Längen- und Breitengrade sowie gegebenenfalls die Höhen der einzelnen Knoten der gefundenen Route in der Reihenfolge, wie sie passiert werden müssen. Dabei werden auch *Pillar Nodes* berücksichtigt.

### 6.3.1 Hinzufügen der Etagenwerte zum Routingergebnis

Da aus einer reinen Liste von Koordinaten nicht erkennbar ist, welche Etagen der Benutzer überqueren muss, um sein Ziel zu erreichen, wurde die Klasse *PointListIndoor* implementiert. Sie leitet sich von der Klasse *PointList* ab und speichert zusätzlich die Werte der Etagen für jeden einzelnen Punkt.

Die Klasse *Path* enthält ebenfalls die Methode *calcEdges*, die die gespeicherten Kanten des Graphen in der Reihenfolge zurückgibt, wie sie passiert werden müssen, um das Ziel zu erreichen. Bei diesen Kanten kann es sich auch um virtuelle Kanten handeln. Eine virtuelle Kante ist eine Kante, die im Navigationsgraph nicht gespeichert ist, weil es sich bei einem der beiden Knoten um einen angenäherten Punkt handelt, der sich auf einer Kante befindet (vgl. Abb. 27). Für eine virtuelle Kante können ebenfalls die *Pillar Nodes* bestimmt werden.

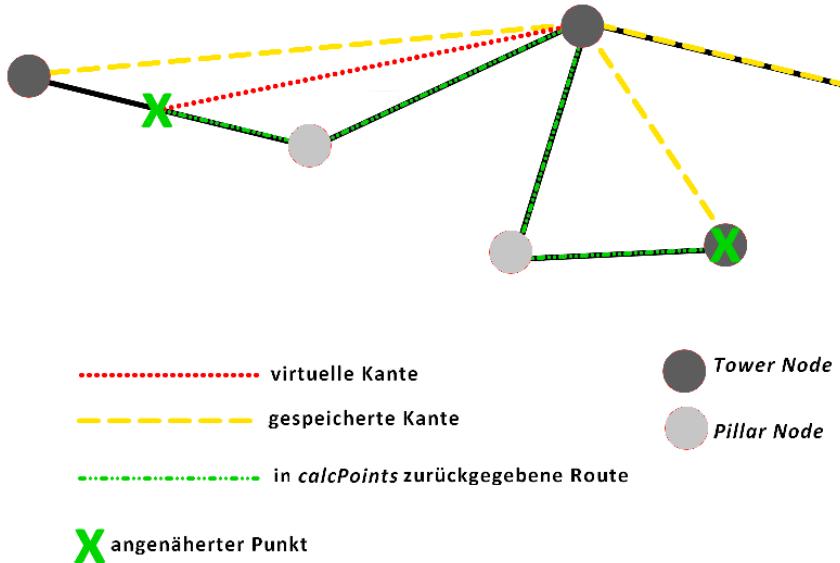


Abbildung 27: Virtuelle Kante im Navigationsgraph

Unter der Voraussetzung, dass es sich um ein Indoor-Routing handelt, werden in der Klasse `Path` nun die Etagenwerte des Ausgangs- und des Zielpunkts gespeichert. Die Methode `calcPoints` wurde so ergänzt, dass sie im Fall eines Indoor-Routings eine Instanz der Klasse `PointListIndoor` statt einer Instanz der Klasse `PointList` zurückgibt. Die Instanz der Klasse `PointList`, die bisher als Ergebnis der Methode zurückgeliefert wurde, wird dabei gemeinsam mit den in `calcEdges` berechneten Kanten und den Etagenwerten des Ausgangs- und des Zielpunkts benötigt, um eine Instanz der Klasse `PointListIndoor` zu erstellen. Nachfolgend wird erläutert, wie die Etagenwerte für die einzelnen Punkte der berechneten Route bestimmt werden (vgl. Abb. 28).

Alle Kanten des ermittelten Pfads werden der Reihe nach betrachtet. Für den ersten Knoten einer Kante wird der Wert der Etage ermittelt. Handelt es sich um eine Kante, die im gespeicherten Graph vorhanden ist, wird der Wert über die erwähnte Grapherweiterung bestimmt. Im Fall einer virtuellen Kante gibt es drei Möglichkeiten. Handelt es sich um die erste Kante des Pfads, erhält der Knoten den Wert der Etage des Ausgangspunkts. Ist die Kante die letzte Kante des Pfads, erhält der Knoten den Wert der Etage des Zielpunkts. Andernfalls erhält der Knoten den Etagenwert, der dem vorhergehenden Knoten zugewiesen wurde. Die *Pillar Nodes*, die auf den jeweils ersten Knoten einer Kante folgen, erhalten immer denselben Etagenwert wie dieser. Als Letztes erhält der letzte Knoten des Pfads den Wert der Etage, der für den Zielpunkt gespeichert wurde.

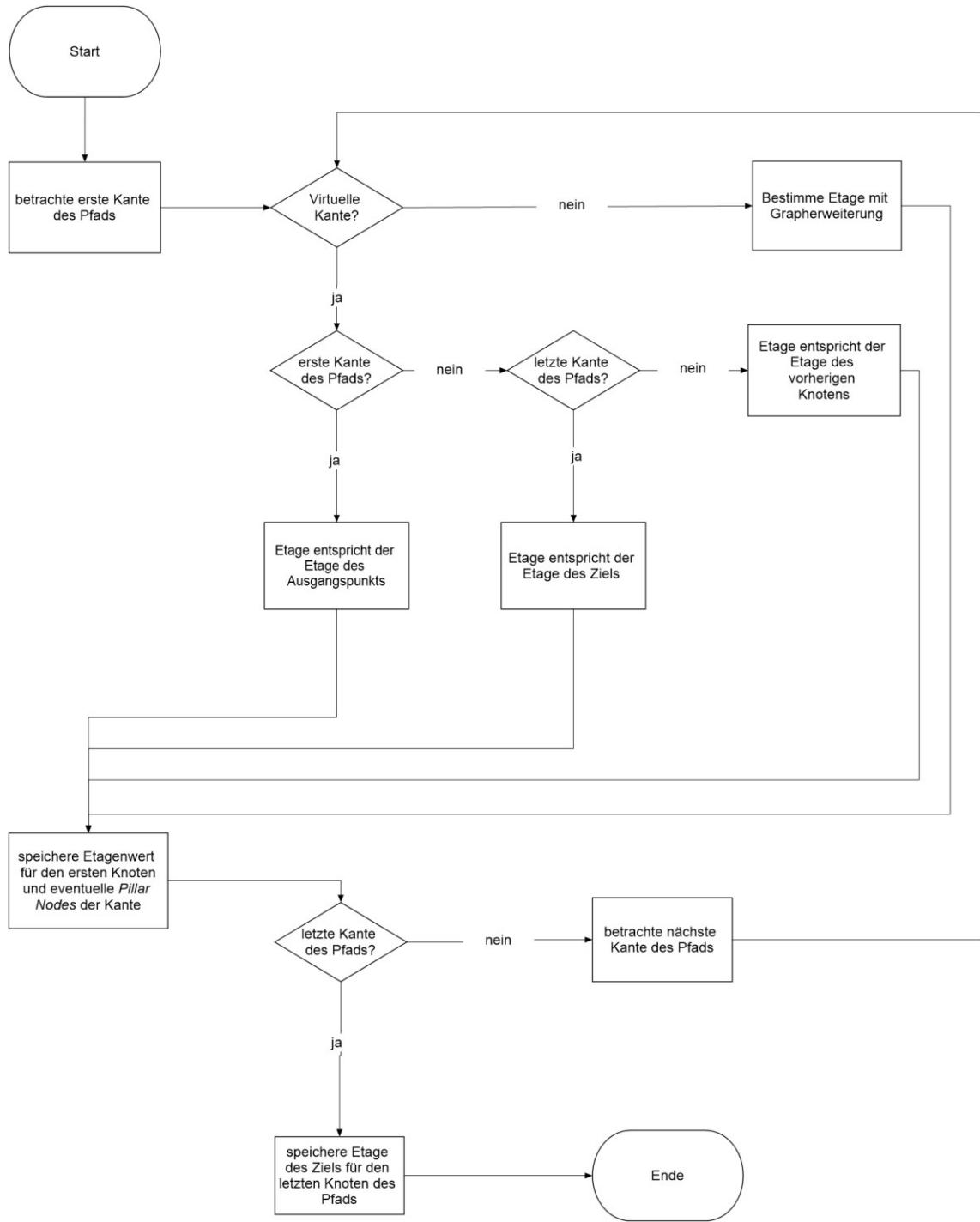


Abbildung 28: Funktionsweise des Algorithmus zur Bestimmung der Etagenwerte der einzelnen Knoten des Pfades

## 6.4 Graphische Darstellung der Routen unter *GraphHopper*

*GraphHopper* bietet auch graphische Oberflächen an, mit denen Suchanfragen erstellt und die gefundenen Routen dargestellt werden können. Eingegangen wird hier auf die Darstellungsweise des Debugging-Tools *MiniGraphUI* und die der Serveranwendung *GHSserver*. Sie wurden jeweils für das Indoor-Routing angepasst. Der Fokus lag dabei auf der Funktionalität, da die Routen langfristig im erwähnten Framework dargestellt werden sollen.

### 6.4.1 *MiniGraphUI*

Innerhalb der Debugging-Anwendung *MiniGraphUI* kann nur ein *Flag Encoder* verwendet werden. Es wird der Navigationsgraph für eine ausgewählte Region angezeigt (vgl. Abb. 29). Die einzelnen Abschnitte des Graphen erhalten unterschiedliche Farben, je nachdem, welche Höchstgeschwindigkeit auf der jeweiligen Straße gilt. Oberhalb des angezeigten Graphen befindet sich eine Legende, die erläutert, welche Farben für welche Geschwindigkeitsbegrenzungen stehen. Bei Bedarf kann der Anwender in den Navigationsgraphen hineinzoomen. Sobald er auf zwei verschiedene Stellen innerhalb des Graphen geklickt hat, wird die Route zwischen den angenäherten Punkten in Rot eingezeichnet (vgl. Abb. 30). Die Knoten des Graphen, die bei der Suche nach der optimalen Route passiert wurden, werden gelb angezeigt.

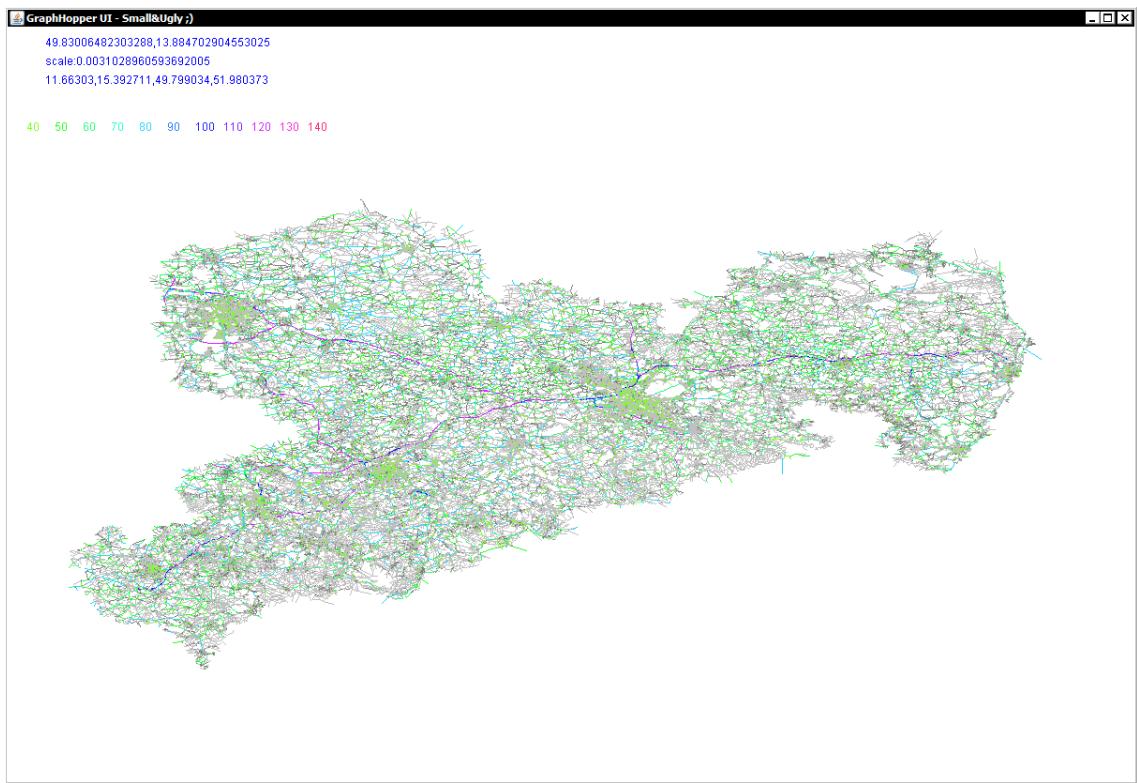


Abbildung 29: Navigationsgraph für Autos für die Region Sachsen unter MiniGraphUI

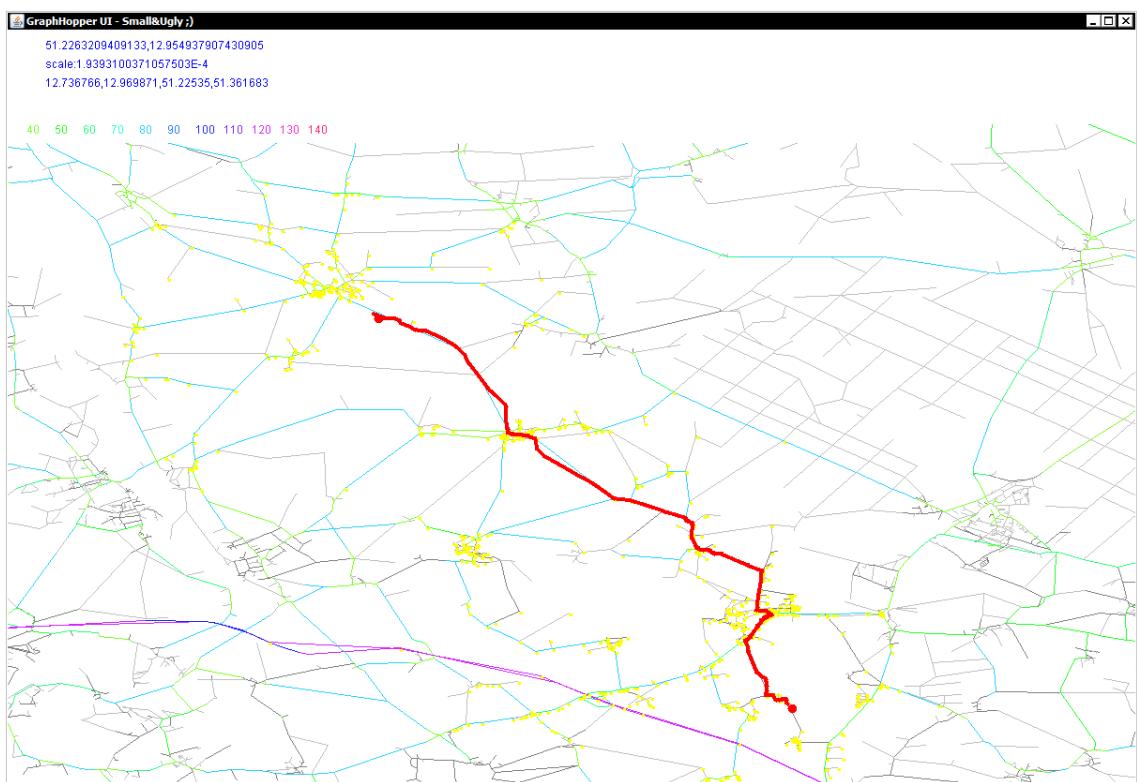


Abbildung 30: Anzeige einer Route für Autos unter MiniGraphUI

#### 6.4.2 *MiniGraphUIIndoor*

Da es sich bei *MiniGraphUI* um eine Debugging-Anwendung handelt, wurde bei der Implementierung kein Wert darauf gelegt, eine spätere Ableitung zu ermöglichen. Aus diesem Grund erschien es leichter, den Quellcode der Anwendung zu kopieren und als *MiniGraphUIIndoor* zu überarbeiten.

Die grundlegende Optik von *MiniGraphUI* wurde beibehalten. Da es innerhalb von Gebäuden keine Geschwindigkeitsbegrenzung gibt, wurde die Legende entfernt. An der Seite wurde ein Etagenselektor hinzugefügt. Des Weiteren wurde eine Gruppe an Buttons hinzugefügt, mit denen zwischen den drei implementierten *Flag Encodern* gewechselt werden kann.

In Abbildung 31 ist die Darstellung eines Navigationsgraphen für Fußgänger des NHGs in *MiniGraphUI* zu sehen. In Abbildung 32 wird derselbe Graph innerhalb von *MiniGraphUIIndoor* dargestellt. Bei den grünen Linien handelt es sich um Wege, die sich auf der Etage befinden, die im Etagenselektor ausgewählt wurde. Die hellgrauen Kanten stellen Kanten des Graphen dar, die sich auf einer anderen Etage als der ausgewählten befinden. Bei einem Routing über mehrere Etagen wird der Teil der Route, der sich auf der aktuell ausgewählten Etage befindet, rot angezeigt, der andere Teil dunkelgrau (vgl. Abb. 33).

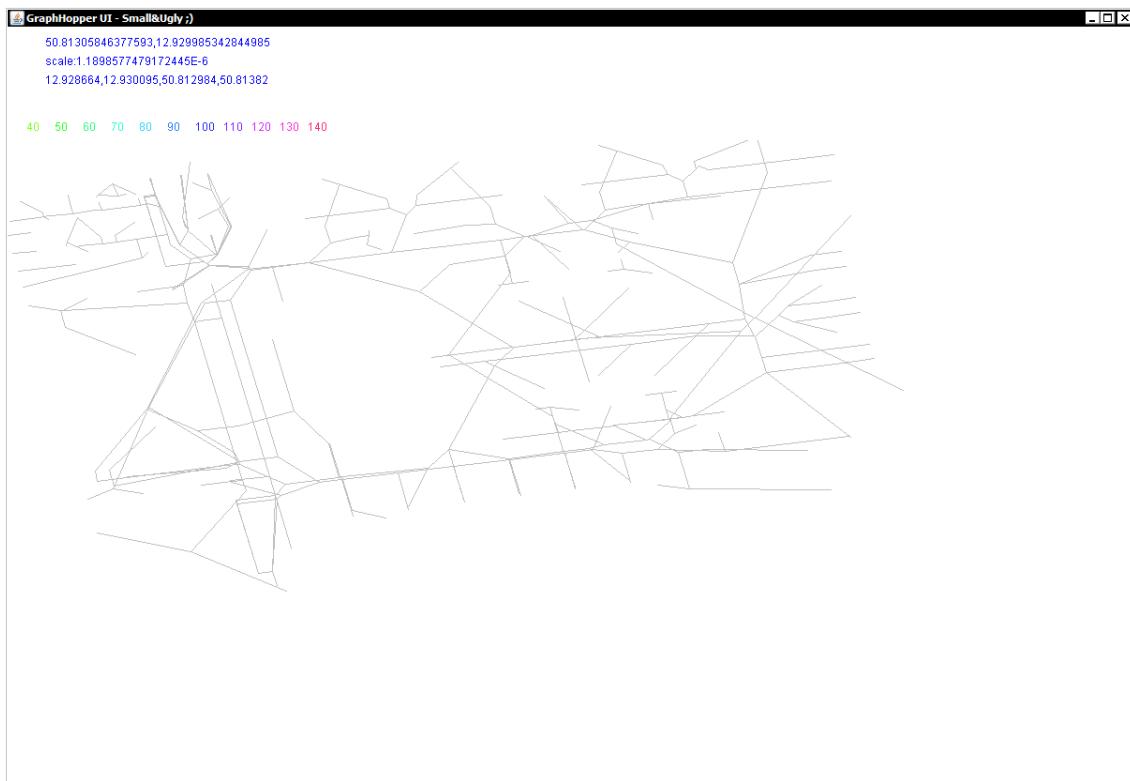


Abbildung 31: Darstellung des Navigationsgraphen für Fußgänger des NHGs unter MiniGraphUI

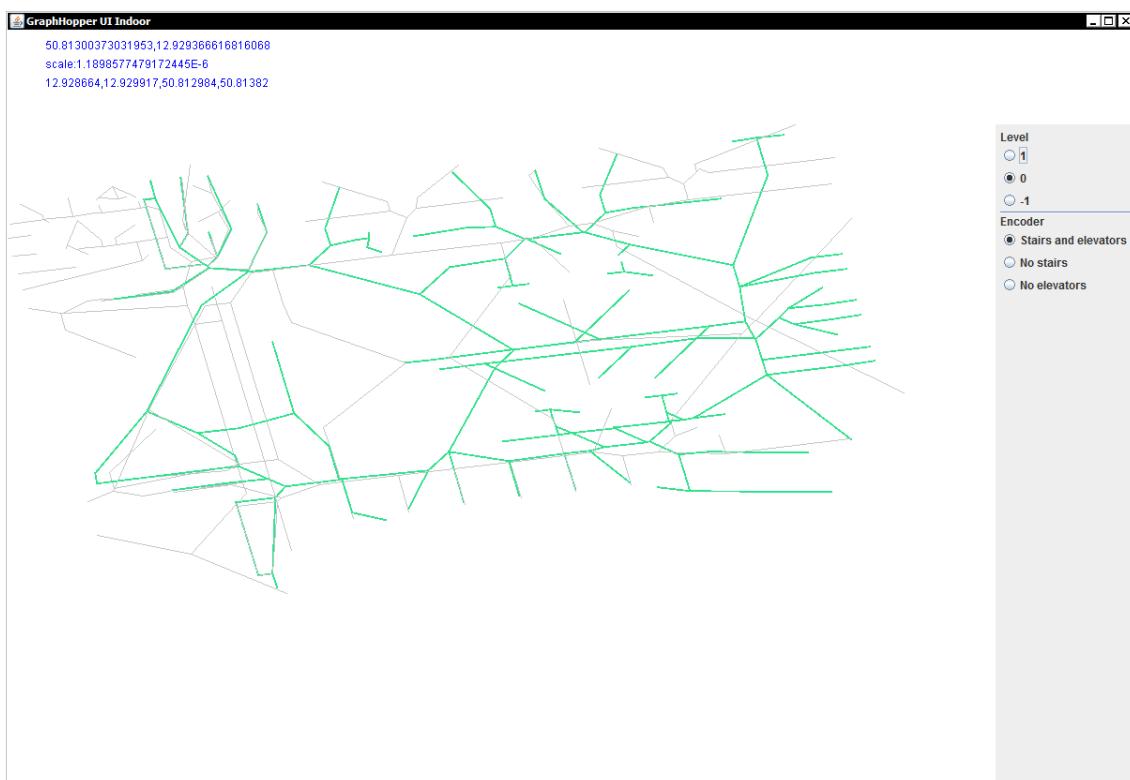


Abbildung 32: Darstellung des Navigationsgraphen des NHGs für die drei implementierten Flag Encoder unter MiniGraphUIIndoor

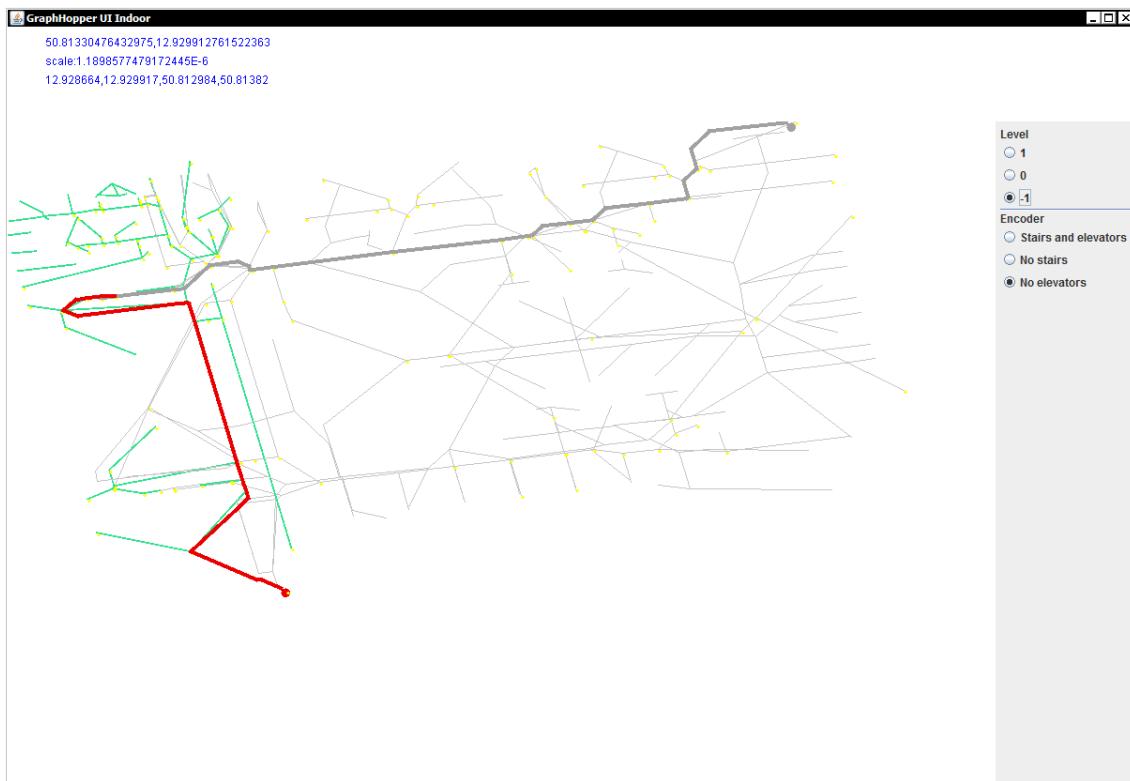


Abbildung 33: Darstellung einer Route vom ersten Obergeschoss ins Untergeschoss unter MiniGraphUIIndoor

#### 6.4.3 GHServer

*GHServer* ermöglicht ein Routing innerhalb einer Webanwendung. Als Grundlage dient dabei der Navigationsgraph einer bestimmten Region. Es können beliebig viele *Flag Encoder* verwendet werden. Den größten Teil der Anwendung nimmt eine skalierbare Karte ein (vgl. Abb. 34). Bei bestehender Internetverbindung kann zwischen Kartendarstellungen verschiedener Server ausgewählt werden. Dargestellt werden dabei nur die Daten, die auf der Serverseite vorliegen. Es können auch eigene Kartenserver verwendet werden. Die Region, für die ein Navigationsgraph übergeben wurde, wird eingehaumt. Links der Kartendarstellung befindet sich eine Eingabemaske. Falls der Navigationsgraph für unterschiedliche Verkehrsmittel übergeben wurde, kann über Buttons oberhalb der Eingabemaske zwischen diesen gewechselt werden.

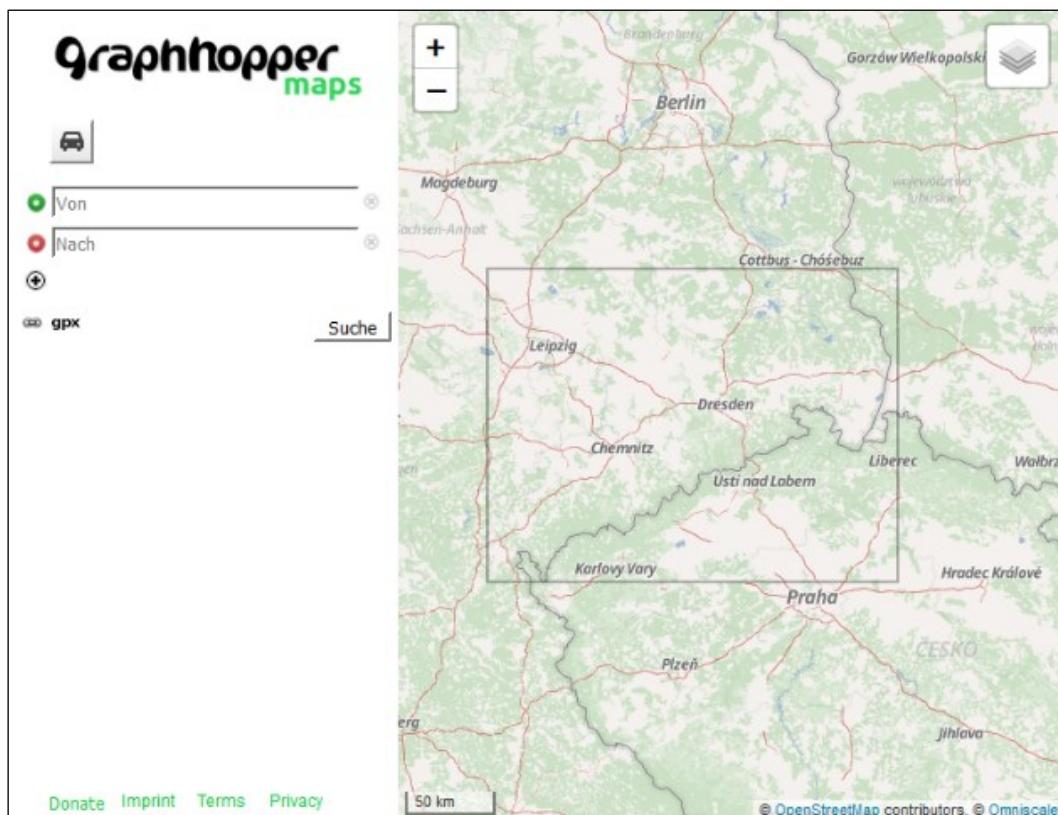


Abbildung 34: Darstellung der Region Sachsen unter GHServer

Der Benutzer kann mit einem Rechtsklick auf die Karte einen Startpunkt, einen Zielpunkt und beliebig viele Zwischenziele auswählen. Die Symbole, die die einzelnen Ziele darstellen, können nachträglich per Drag and Drop verschoben werden. Alternativ können in die Eingabemaske beliebig viele Punkte in Form von Koordinaten eingegeben werden (vgl. Abb. 35). Sobald mindestens zwei Punkte auf der Karte ausgewählt wurden oder die Eingabe in der Eingabemaske bestätigt wurde, wird per JavaScript eine Suchanfrage abgeschickt, die von einem Servlet beantwortet wird. Wurde eine Route gefunden, wird diese farbig dargestellt und unterhalb der Eingabemaske eine Wegbeschreibung angezeigt.

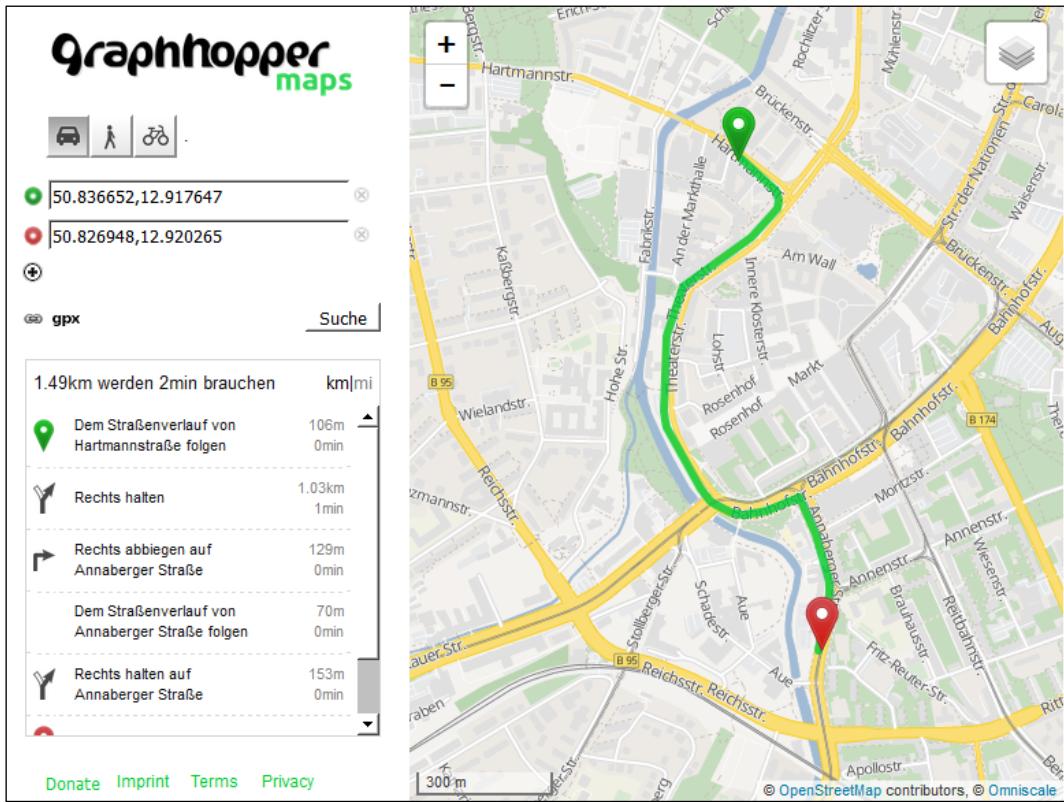


Abbildung 35: Darstellung einer Route im Chemnitzer Stadtgebiet unter GHServer

#### 6.4.4 Erweiterung von *GHServer* für den Indoor-Bereich

Die Indoor-Daten des NHG sind zum Zeitpunkt des Schreibens noch nicht auf dem öffentlichen OSM-Server zu finden. Da bei der Darstellung der skalierbaren Karte auf diese öffentlichen Daten zurückgegriffen wird, wird das gesamte Gebäude in der Darstellung von *GHServer* als graue Fläche dargestellt (vgl. Abb. 36).

Für die Darstellung eines Indoor-Routings wurde zunächst die HTML-Darstellung angepasst. Unterhalb der Eingabemaske wurde ein Etagenselektor hinzugefügt. Dieser wurde dem NHG angepasst, sodass das Untergeschoss, das Erdgeschoss und das erste Stockwerk als aktuelle Etage ausgewählt werden können. In Zukunft sollte der Etagenselektor dynamisch gestaltet sein, um abhängig vom Gebäude mehr oder weniger Auswahlmöglichkeiten zur Verfügung zu stellen.

Bei der Übergabe der Punkte muss zusätzlich zu den Koordinaten nun auch ein Etagenwert übergeben werden. Mit Hilfe des Etagenselektors kann der Benutzer zwischen den Ansichten der einzelnen Etagen wechseln. Wie zuvor kann er einen Punkt mit einem Rechtsklick auf der Karte auswählen. Als Etagenwert wird automatisch der Wert der Etage hinzugefügt, die zurzeit im Etagenselektor ausgewählt ist. Es ist auch möglich, die

Koordinaten eines Punkts direkt über die Eingabemaske einzugeben. Dabei muss der Etagenwert mit einem Komma den Koordinaten angefügt werden (zum Beispiel 50.81578448141, 12.93174512204, -1). Eine Suche nach Raumnamen und die automatische Ermittlung der entsprechenden Koordinaten sind derzeit noch nicht möglich.

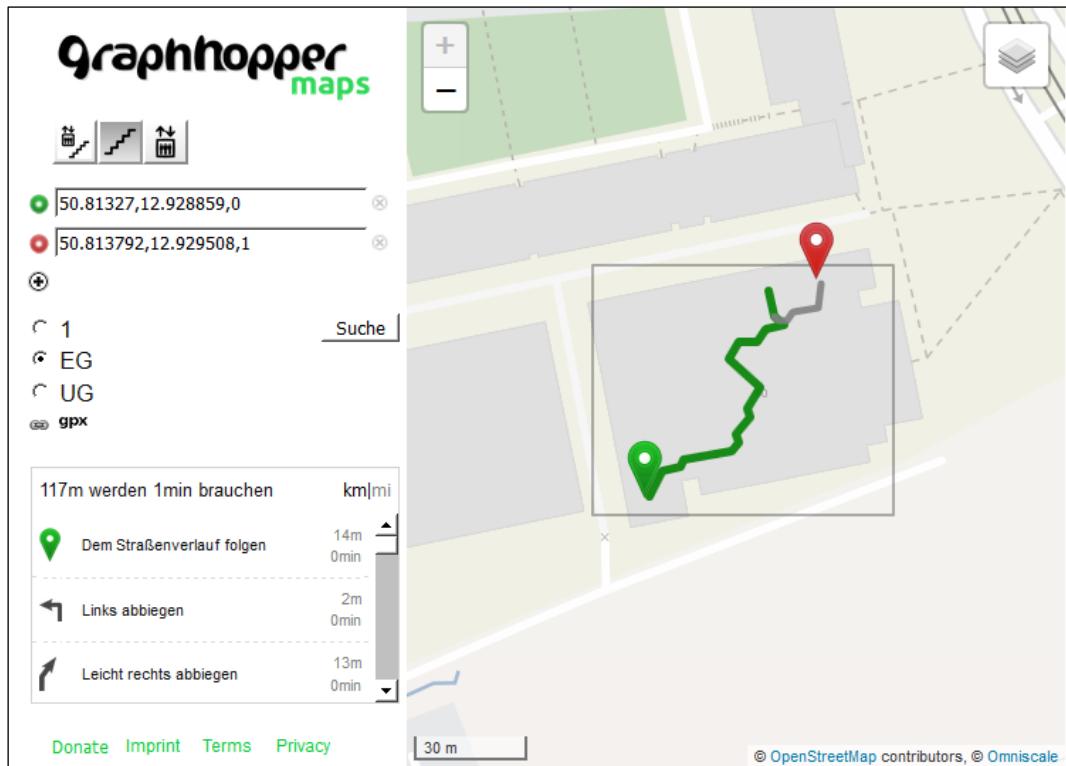


Abbildung 36: Darstellung einer Route zwischen dem Erdgeschoss und dem ersten Obergeschoss bei Vermeidung von Fahrstühlen  
Aktuell ausgewählte Etage: Erdgeschoss

Das Servlet, das die JavaScript-Anfrage entgegennimmt, wurde so erweitert, dass es im Falle eines Indoor-Routings nur Punkte akzeptiert, die einen Etagenwert enthalten. Wurde eine Route zwischen den Punkten gefunden, gibt das Servlet die einzelnen Punkte, über die das Ziel erreicht werden kann, inklusive der jeweiligen Etagenwerte als Antwort zurück. Die Abschnitte der Route, die sich auf der aktuell ausgewählten Etage befinden, werden grün dargestellt. Der Rest der Route wird grau eingefärbt.

In Abbildung 36 ist eine Route zwischen einem Punkt im Erdgeschoss und einem Punkt im 1. Obergeschoss zu sehen. Es wurde dabei der *Flag Encoder* ausgewählt, der Wege ausschließt, die über Fahrstühle führen. In Abbildung 37 ist die Route zu sehen, die zwischen denselben Punkten berechnet wird, wenn der *Flag Encoder* ausgewählt wird, der

Wege über Treppen ausschließt. Neben der abweichenden Routenführung ist zu erkennen, wie sich die Farbdarstellung je nach ausgewählter Etage ändert.

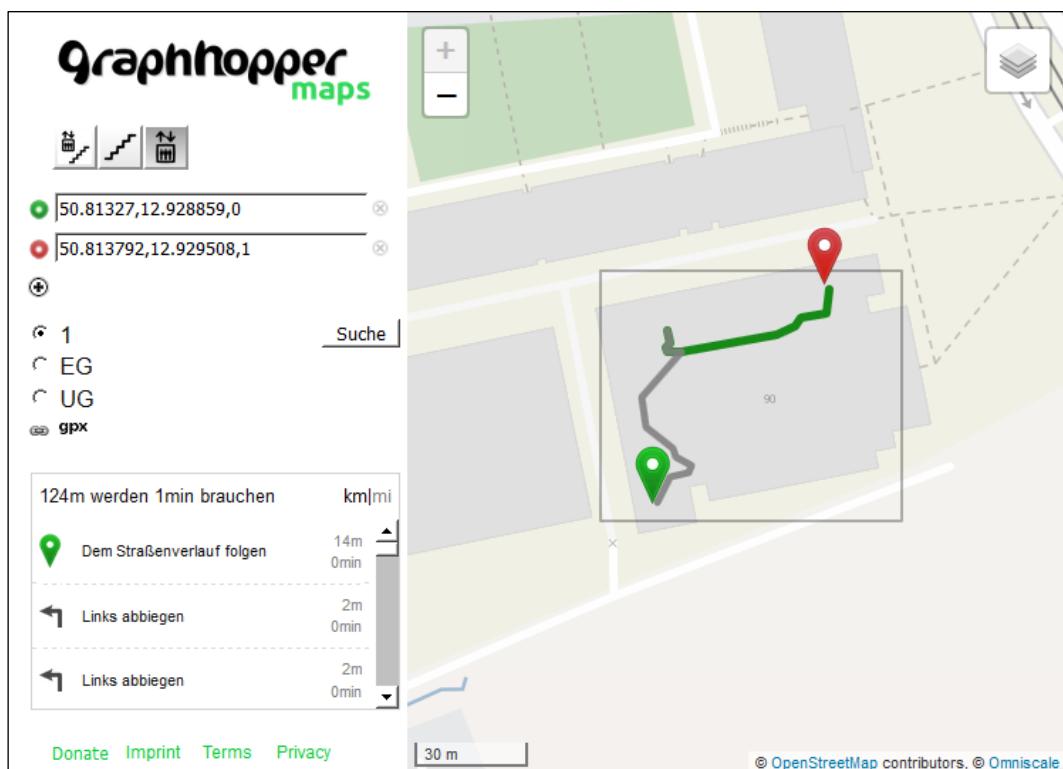


Abbildung 37: Darstellung der Route zwischen den Punkten aus Abb. 33 bei Vermeidung von Treppen  
Aktuell ausgewählte Etage: Erstes Obergeschoss

## 7 Nutzertest

Um die Ergebnisse dieser Arbeit besser bewerten zu können, wurde ein Nutzertest durchgeführt. Dieser bestand aus einem Zeichenteil und einer Befragung der Teilnehmer.

Im Zeichenteil sollten die Teilnehmer in Anlehnung an [WY15] Routen zwischen ausgewählten Punkten in einen Gebäudeplan des NHGs einzeichnen. Folgende Fragen sollten damit beantwortet werden:

1. Möchten die Teilnehmer Wege innerhalb einer Indoor-Routing-Anwendung so dargestellt bekommen, wie sie sie auch gehen würden?
2. Inwieweit unterscheiden sich die gezeichneten Routen der Teilnehmer von den durch *GraphHopper* berechneten?
3. Mit Hilfe welcher optionalen Funktionen des Tools zur automatischen Wegenetzwerksgenerierung werden die besten Wegenetzwerke gebildet?

In der anschließenden Befragung wurde zum einen auf Besonderheiten in den Zeichnungen der Teilnehmer eingegangen. Zum anderen sollte herausgefunden werden, ob bei den Teilnehmern ein grundsätzliches Interesse an einer Indoor-Routing-Anwendung besteht und welche Funktionen sie in diesem Fall von solch einer Anwendung erwarten würden.

Insgesamt nahmen zehn Personen am Nutzertest teil. Die Anzahl der männlichen und weiblichen Teilnehmer war ausgeglichen. Vier der Teilnehmer kannten das NHG nicht, während die anderen sechs dort bereits häufiger waren. Sechs der Teilnehmer gaben an, sich eher schlecht in Gebäuden orientieren zu können. Die anderen vier schätzten ihre Orientierung innerhalb von Gebäuden eher gut ein.

Im Folgenden werden der Zeichenteil und die Befragung genauer vorgestellt und die jeweiligen Ergebnisse zusammengefasst.

### 7.1 Zeichenteil

Der Zeichenteil war ebenfalls in zwei Teile untergliedert. Im ersten Teil sollten die Teilnehmer zwischen jeweils zwei vorgegebenen Punkten im Gebäudeplan den Weg einzeichnen, den sie gehen würden, um das Ziel möglichst schnell zu erreichen. Es gab insgesamt fünf verschiedene Zeichenaufgaben, die jeder Teilnehmer bearbeiten musste.

Im zweiten Teil wurden die Teilnehmer gefragt, ob sie innerhalb einer Indoor-Routing-Anwendung eine andere Darstellung der Wege bevorzugen würden. In diesem Fall sollten sie die Wege erneut in einer anderen Farbe einzeichnen.

Die Start- und Endpunkte jeder Aufgabe wurden ohne Betrachtung der generierten Wegenetzwerke ausgesucht. Es wurde versucht, die Punkte so auszuwählen, dass es mehrere mögliche Verbindungen zwischen ihnen gibt. Bei drei der Aufgaben befinden sich Start- und Endpunkt auf derselben Etage. In den anderen beiden Aufgaben befindet sich je ein Punkt auf der ersten und ein Punkt auf der zweiten Etage.

Den Teilnehmern wurde ausdrücklich mitgeteilt, dass jeder Raum leer angesehen und durchquert werden kann. Bei den Aufgaben, in denen sich Start und Ziel auf verschiedenen Etagen befinden, wurden die Teilnehmer aufgefordert, eine Verbindung über Treppen<sup>11</sup> zu wählen. Dabei sollten sie jeweils nur den Weg zu einer Treppe hin und von einer Treppe weg zeichnen. Als Hilfestellung wurden die Treppen im Gebäudeplan farblich hervorgehoben.

Mit Hilfe von *GHServer* wurden die Koordinaten der einzelnen Aufgaben an *GraphHopper* übergeben. Für jede Aufgabe wurden auf der Grundlage vier automatisch generierter Wegenetzwerke unterschiedliche Routen berechnet. Die zugrundeliegenden Wegenetzwerke wurden alle mit Hilfe des *Straight Skeletons* und der Funktion *add\_supplementary\_ways* gebildet. Ansonsten unterschieden sich die Wegenetzwerke wie folgt:

- Wegenetzwerk 1: mit keiner weiteren Funktion ergänzt oder überarbeitet
- Wegenetzwerk 2: überarbeitet mit *simplify\_ways*
- Wegenetzwerk 3: ergänzt mit *door\_to\_door*
- Wegenetzwerk 4: überarbeitet und ergänzt mit *simplify\_ways* und *door\_to\_door*

Im folgenden Abschnitt werden die fünf Zeichenaufgaben einzeln vorgestellt und die Gründe für die Auswahl der jeweiligen Punktpaare erläutert. Anschließend werden jeweils die gezeichneten Routen der Teilnehmer mit den durch *GraphHopper* berechneten Routen verglichen. Ähneln sich bei der Anwendung von *GraphHopper* auf unterschied-

---

<sup>11</sup> Die Wahl von Routen über Fahrstühle wurde nicht getestet, weil sich innerhalb des NHGs nur ein Fahrstuhl befindet und sich die Routen der einzelnen Teilnehmer dadurch zwangsweise geähnelt hätten.

lich generierte Wegenetzwerke mehrere Routen stark, wird nur eine dieser Routen abgebildet. Die gesamten Ergebnisse unter *GraphHopper* sowie die einzelnen Bearbeitungen der Teilnehmer sind im Anhang zu finden.

### 7.1.1 Zeichenaufgabe 1

Start und Ziel in der ersten Aufgabe befinden sich jeweils auf der ersten Etage. Der eine Punkt befindet sich in der Mitte eines Seminarraumes, der andere in einem Korridor unmittelbar vor dem Eingang zu einem Hörsaal. Dieses Punktpaar wurde gewählt, um herauszufinden, ob die Teilnehmer den kürzesten Weg durch den Hörsaal wählen würden.

Bis auf eine Person entschieden sich alle Teilnehmer für eine direkte Route durch den Hörsaal (vgl. Abb. 38<sup>12</sup>). Die Person, die sich gegen eine Route durch den Hörsaal entschieden hatte, gab an, voreingenommen gewesen zu sein, weil sie das Gebäude kannte. Sonst hätte sie auch den Weg durch den Hörsaal gewählt. Die grundsätzliche Routendarstellung der anderen neun Teilnehmer ähnelt sich.

Sechs Teilnehmer zeichneten für die Anzeige innerhalb einer Anwendung Wege, die den ursprünglich gezeichneten ähnelten, jedoch statt Kurven Ecken enthielten. Eine Testperson entschied sich dafür, die Route nun um den Hörsaal herumzuführen. Die restlichen drei Teilnehmer entschieden sich dagegen, die Routen im zweiten Teil der Aufgabe anzupassen.

Bei der Bearbeitung der Aufgabe mit *GraphHopper* wird bei dem Wegenetzwerk, das ohne die Anwendung zusätzlicher Funktionen generiert wurde, eine Route berechnet, die sich stark von denen der Teilnehmer unterscheidet (vgl. Abb. 39 links). Sie führt durch einen anderen Hörsaal und enthält außerdem einen Umweg zu einer Tür, die in einen Nebenraum dieses Hörsaals führt. Die Routen, die in den anderen drei Fällen generiert wurden, entsprechen ähnlich denen der Teilnehmer (vgl. Abb. 39 rechts). Grundsätzlich entsprechen sie eher den Routen, die die Teilnehmer im ersten Teil zeichneten, weil sie eher kurvenartig verlaufen.

---

<sup>12</sup> Ein Teilnehmer entschied sich dafür, die Routen für die Darstellung innerhalb einer Anwendung deutlich breiter zu zeichnen. Seine Bearbeitungen der Aufgaben werden deshalb jeweils gesondert gezeigt, um nicht die Ergebnisse der anderen Teilnehmer zu verdecken.



Abbildung 38: Bearbeitungen der ersten Zeichenaufgabe  
Blau: Wege zur schnellsten Zielerreichung, rot: Wege für die Darstellung in einer Routing-Anwendung

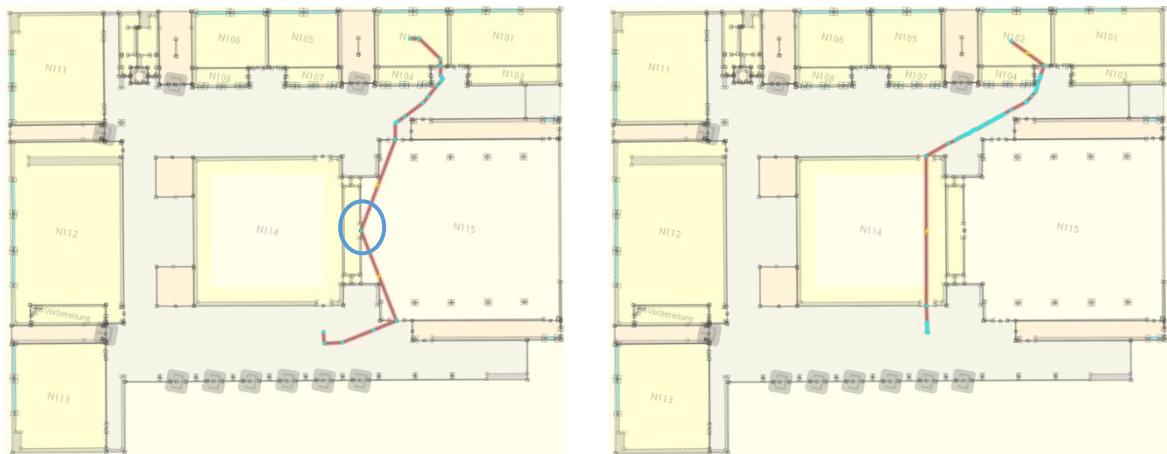


Abbildung 39: Durch GraphHopper berechnete Route für die erste Aufgabe  
Wegenetzwerk generiert durch:  
Links: keine zusätzlichen Funktionen  
Rechts: simplify\_ways und door\_to\_door  
Blau umkreist: Tür, die nicht durchquert wird

### 7.1.2 Zeichenaufgabe 2

In der zweiten Zeichenaufgabe befinden sich Start und Ziel in zwei sich schräg gegenüberliegenden Räumen. Diese Aufgabe wurde gewählt, um zu überprüfen, ob sich die Teilnehmer für die direkte Verbindung zwischen den zwei Türen entscheiden würden.

Mit Ausnahme einer Person zeichneten alle Teilnehmer als schnellste Verbindung, die sie laufen würden, eine Diagonale zwischen den zwei Räumen (vgl. Abb. 40). Diese Diagonale führt zwischen zwei Säulen hindurch. Die zehnte Person entschied sich für einen Weg um die Säulen herum.

Vier Personen zeichneten die Routen für die Darstellung in einer Anwendung um die Säulen herum. Die anderen Teilnehmer sahen keine Notwendigkeit dafür, die Routen anders darzustellen.

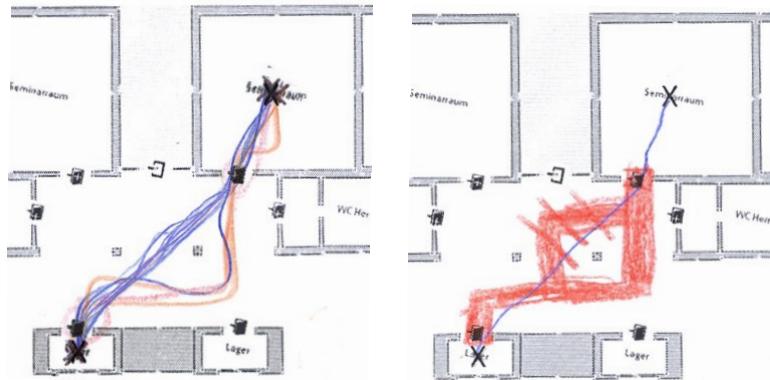


Abbildung 40: Bearbeitungen der zweiten Zeichenaufgabe  
Blau: Wege zur schnellsten Zielerreichung, rot: Wege für die Darstellung in einer Routing-Anwendung

Unter *GraphHopper* werden in allen Fällen Routen gebildet, die zwischen den Säulen hindurchführen (vgl. Abb. 41). Diese ähneln allerdings nur den Routen der Teilnehmer, wenn bei der Generierung des zugrundeliegenden Wegenetzwerks auch die Funktion *door\_to\_door* aufgerufen wurde (vgl. Abb. 41 rechts).

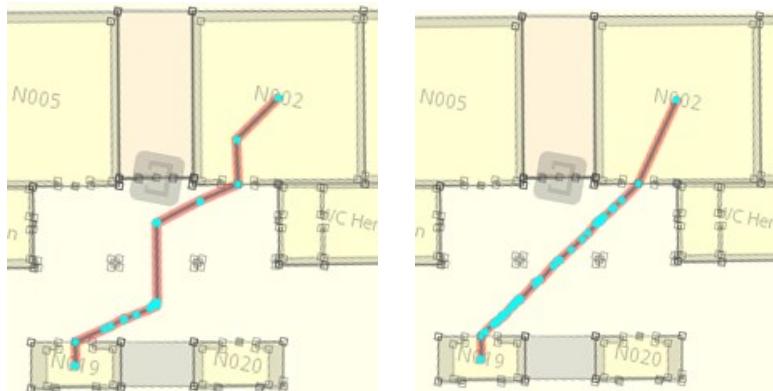


Abbildung 41: Durch GraphHopper berechnete Routen für die zweite Aufgabe  
Links: Wegenetzwerk generiert ohne zusätzliche Funktionen  
Rechts: Wegenetzwerk generiert durch *door\_to\_door* und *simplify\_ways*

### 7.1.3 Zeichenaufgabe 3

Die Punkte der dritten Aufgabe befinden sich jeweils in der Mitte zweier Räume, die sich in entgegengesetzten Ecken des Gebäudes befinden. Um den einen Raum vom anderen aus zu erreichen, muss eine große Fläche überquert werden, auf der sich vier Lagerräume befinden. Jeweils zwei Lagerräume sind durch eine große Bank miteinander verbunden

und bilden dadurch insgesamt zwei Barrieren. Die Punkte dieser Aufgabe wurden ausgewählt, um herauszufinden, welche Art der Umrundung dieser Barrieren die Teilnehmer bevorzugen.

Zwei Personen zeichneten im ersten Teil Wege, die größtenteils parallel zu Wänden verlaufen (vgl. Abb. 42, oben links). Für die Darstellung der Wege innerhalb einer Anwendung passten diese Teilnehmer ihre Zeichnungen nicht an. Die restlichen acht Personen entschieden sich im ersten Teil für Routen, die in einer Kurve mittig zwischen den zwei Barrieren hindurchführen (vgl. Abb. 42 oben rechts und unten). Von diesen Teilnehmern entschieden sich sechs dafür, die Routen für die Darstellung in einer Anwendung anzupassen. Drei Personen passten die Routen nur minimal an, indem sie Kurven durch Geraden ersetzten (vgl. Abb. 42 oben rechts und unten links). Die restlichen drei Teilnehmer zeichneten ebenfalls Routen, die sich an den Wänden orientierten. Die dabei entstandenen Wege unterscheiden sich sehr (vgl. Abb. 42, unten rechts).

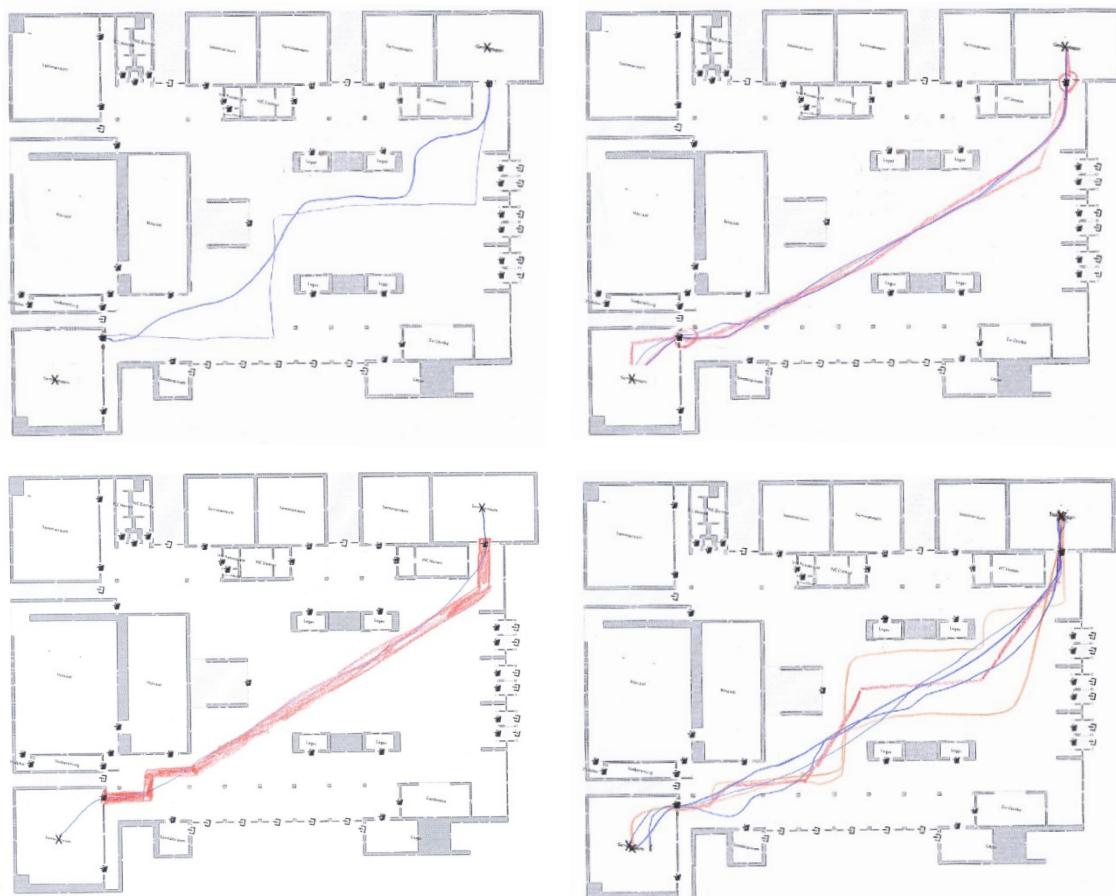


Abbildung 42: Bearbeitungen der dritten Zeichenaufgabe  
Blau: Wege zur schnellsten Zielerreichung, rot: Wege für die Darstellung in einer Routing-Anwendung

Die von *GraphHopper* berechneten Routen führen ebenfalls zwischen den Barrieren hindurch. Bei dem Netzwerk, das ausschließlich durch das *Straight Skeleton* generiert wurde, enthält die Route erneut einen Umweg über eine zusätzliche Tür (vgl. Abb. 43a). Auch das Netzwerk, das ausschließlich durch die Ergänzung mit dem *Door-to-Door*-Algorithmus gebildet wurde, liefert eine Route, die denen der Teilnehmer nur ansatzweise ähnelt (vgl. Abb. 43b). Wenn das Wegenetzwerk zuvor mit *simplify\_ways* überarbeitet wurde, entstehen Routen, die am ehesten denen der Teilnehmer gleichen (vgl. Abb. 43c und 43d).

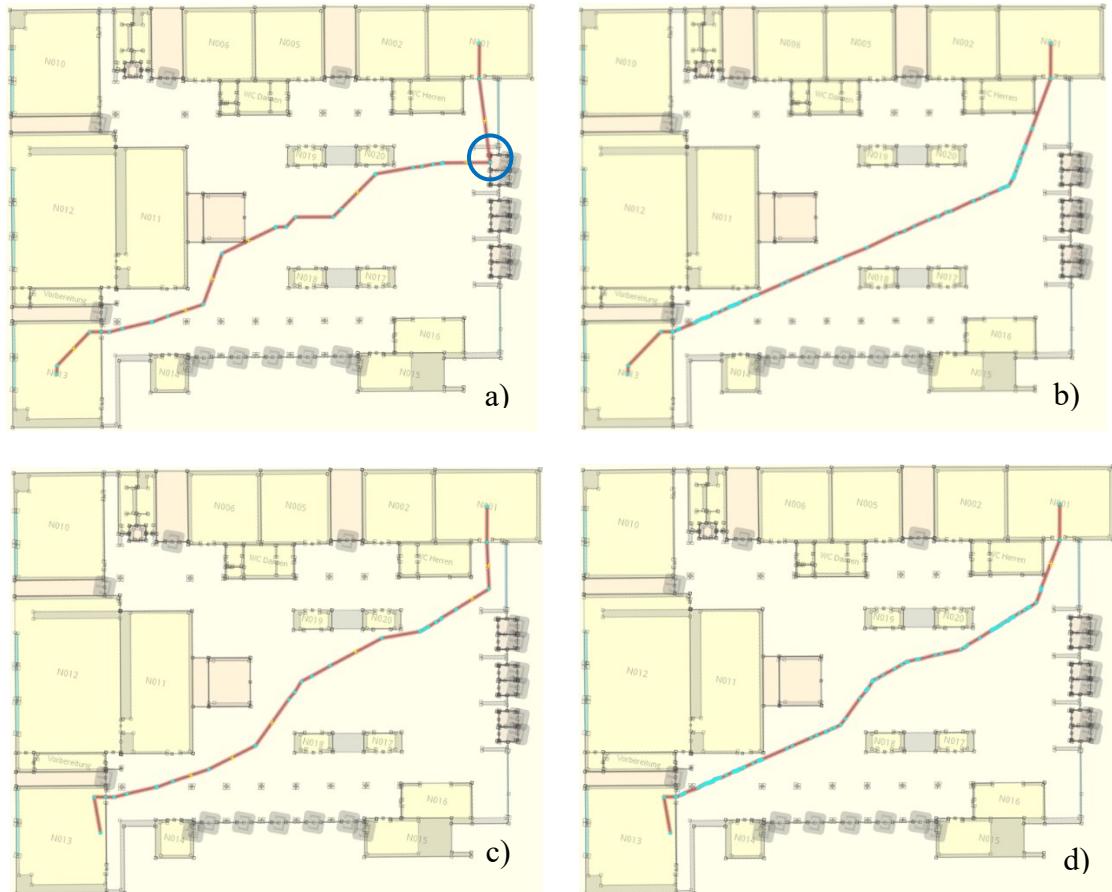


Abbildung 43: Durch GraphHopper berechnete Routen für die dritte Aufgabe

Wegenetzwerk generiert durch:

- a) keine zusätzlichen Funktionen
  - b) door\_to\_door
  - c) simplify\_ways
  - d) simplify\_ways und door\_to\_door
- Blau umkreist: Tür, die nicht durchquert wird

#### 7.1.4 Zeichenaufgaben 4 und 5

In den letzten beiden Aufgaben befinden sich jeweils ein Punkt im Erdgeschoss und einer im ersten Obergeschoss des NHGs. Die Punkte wurden so ausgewählt, dass sich mehrere Treppen in der Nähe befinden. Damit sollte überprüft werden, ob die Teilnehmer bei der

Auswahl der Treppen ähnlich handeln würden. Die Teilnehmer, die das Gebäude kannten, wurden gebeten, sich bei der Auswahl der Routen nicht auf ihre Vorkenntnisse zu stützen.

In der vierten Aufgabe entschieden sich acht Teilnehmer dafür, die Etage über die freistehende Treppe zu wechseln (vgl. Abb. 44.1). Von diesen Teilnehmern entschieden sich drei dafür, die Routen im zweiten Teil anzupassen. Die neu gezeichneten Routen dieser Teilnehmer unterscheiden sich von den ursprünglichen nur dadurch, dass Kurven durch Geraden ersetzt wurden.

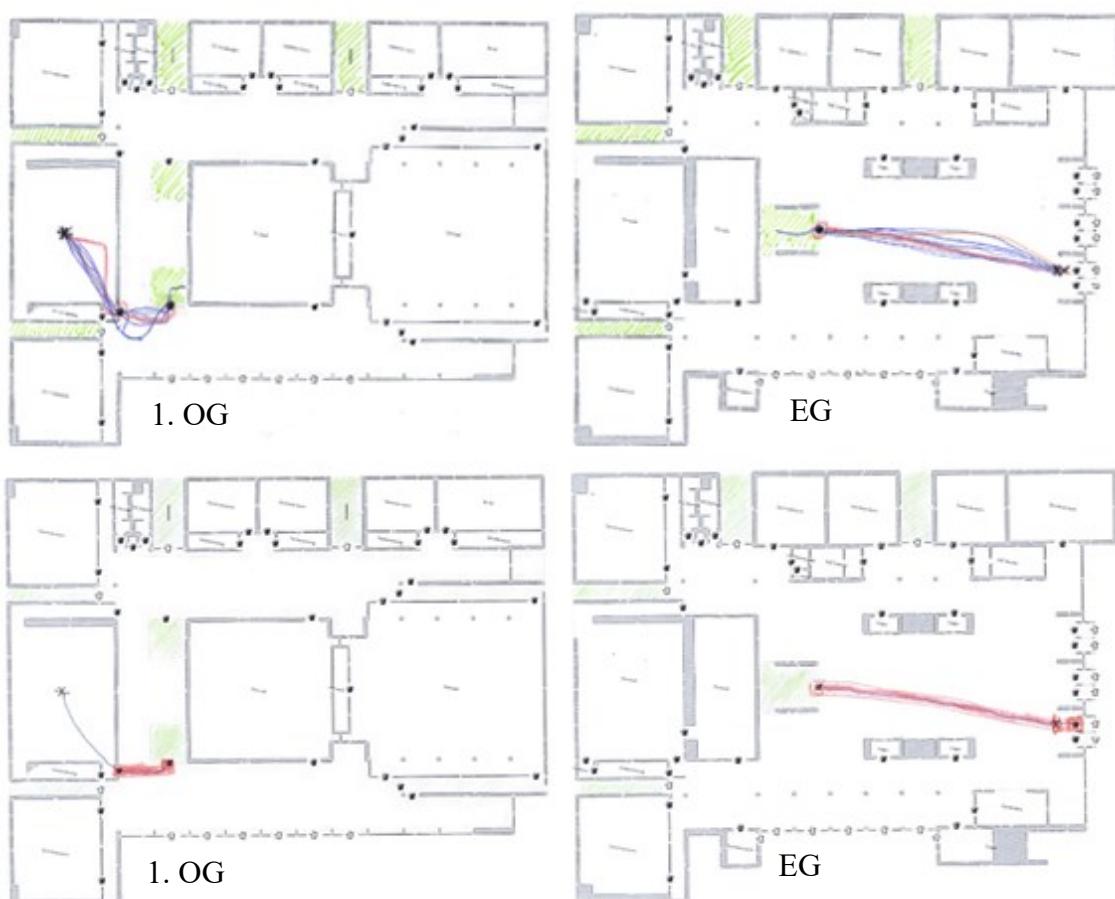


Abbildung 44.1: Bearbeitungen der vierten Zeichenaufgabe  
Blau: Wege zur schnellsten Zielerreichung, rot: Wege für die Darstellung in einer Routing-Anwendung

Die verbliebenen zwei Teilnehmer wählten eine Verbindung über das Treppenhaus, das neben dem Raum im ersten Obergeschoss liegt, in dem sich einer der beiden Punkte befindet (vgl. Abb. 44.2). Der Weg vom Treppenhaus zum zweiten Punkt unterscheidet sich bei beiden Teilnehmern. Die Routen, die diese beiden Teilnehmer im zweiten Teil zeichneten, enthalten erneut eckige Strukturen anstelle von Kurven.



Abbildung 44.2: Bearbeitungen der vierten Zeichenaufgabe  
Blau: Wege zur schnellsten Zielerreichung, rot: Wege für die Darstellung in einer Routing-Anwendung

Bei der Übergabe der Koordinaten an *GraphHopper* zeigte sich, dass das Gebäude nicht genau genug im OSM-Format dargestellt worden war. In dem Raum in der ersten Etage wurde eine Zwischenwand nicht erfasst. Dies führte dazu, dass der berechnete kürzeste Weg durch diese Wand hindurchführt. Aus demselben Grund führt der Weg auch zu dem anderen Teil der Freitreppe, die die Mehrzahl der Teilnehmer als Verbindung zwischen den zwei Etagen gewählt hatte (vgl. Abb. 45). Die Routen auf dieser Etage gleichen sich bei allen zugrundeliegenden Wegenetzen. Im Erdgeschoss unterscheiden sie sich je nachdem, ob das Wegenetzwerk mit Hilfe des *Door-to-Door*-Algorithmus ergänzt wurde oder nicht. Im ersten Fall führt die Route von der Treppe zum zweiten Punkt an der Wand der einen Barriere entlang (vgl. Abb. 45 oben). Andernfalls führt der Weg direkt zum zweiten Punkt und ähnelt damit der Darstellung der Hälfte der Teilnehmer (vgl. Abb. 45 unten).

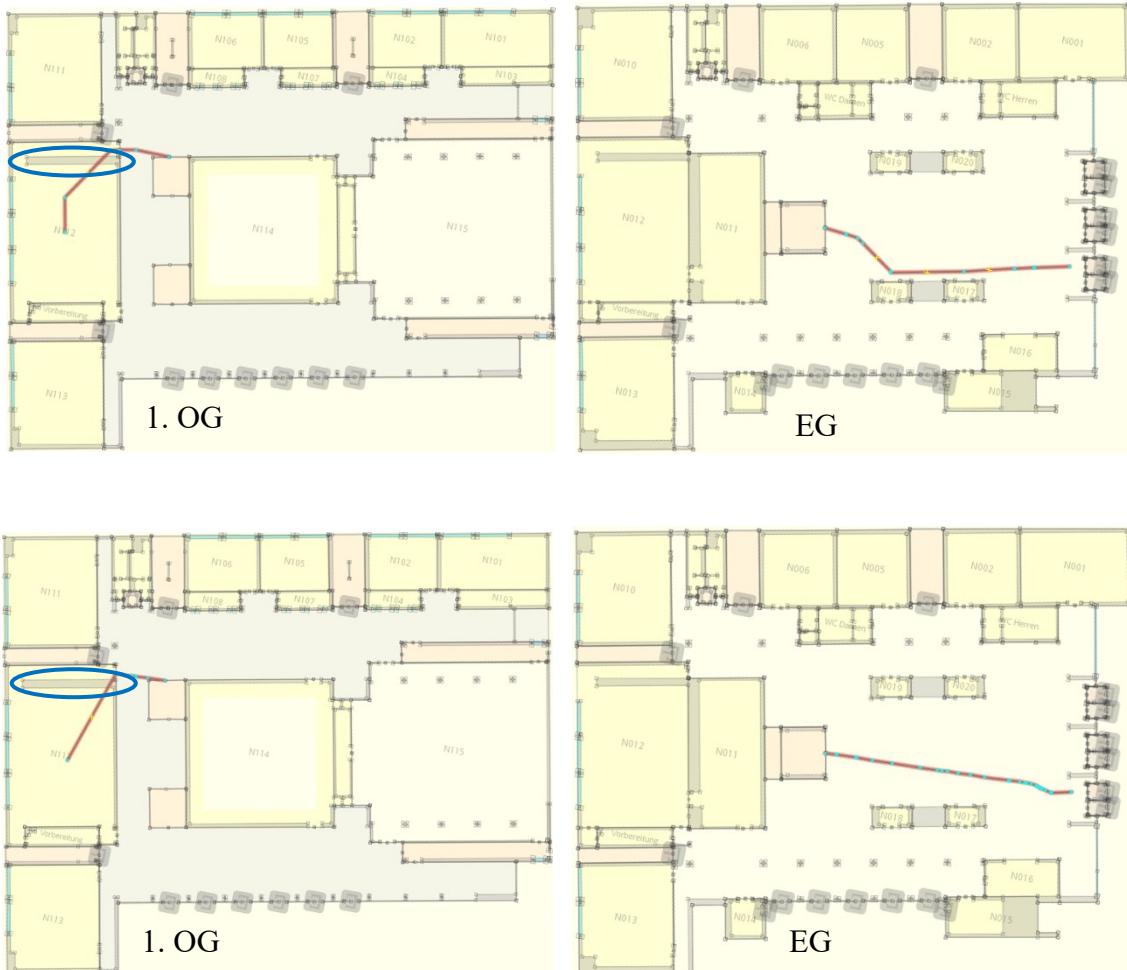


Abbildung 45: Durch GraphHopper berechnete Routen für die vierte Aufgabe  
 Oben: Wegenetzwerk generiert ohne zusätzlichen Funktion  
 Unten: Wegenetzwerk generiert mit simplify\_ways und door\_to\_door  
 Blau umkreist: nicht erfasste Zwischenwand

In der fünften Aufgabe entschied sich die Hälfte der Teilnehmer für eine Route über die freistehende Treppe (vgl. Abb. 46.1). Davon entschieden sich drei Teilnehmer, die Route für die Darstellung innerhalb einer Anwendung neu zu zeichnen. Eine Darstellung unterschied sich dabei nur unwesentlich von den ursprünglich gezeichneten Wegen. In den anderen beiden Darstellungen wurden erneut eckige Strukturen favorisiert. Die Umsetzungen unterschieden sich dabei jedoch.

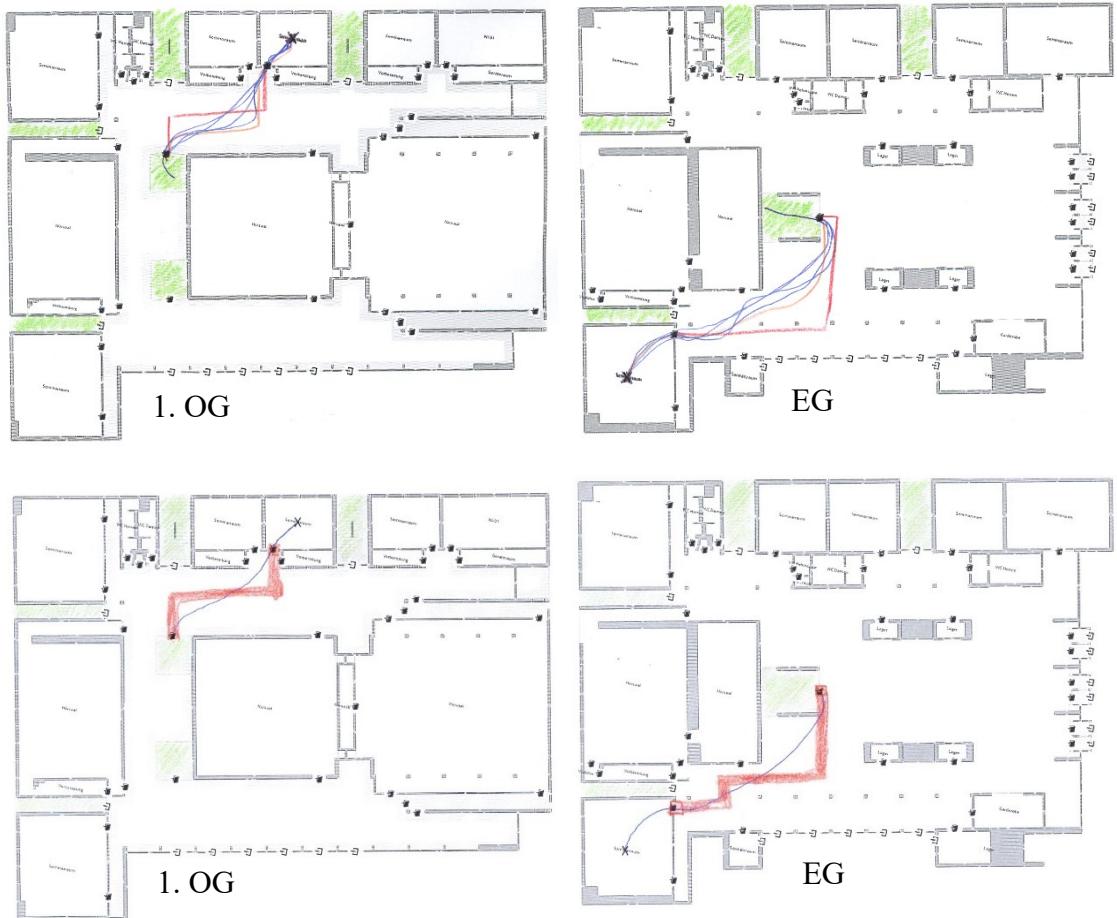


Abbildung 46.1: Bearbeitungen der fünften Zeichenaufgabe  
Blau: Wege zur schnellsten Zielerreichung, rot: Wege für die Darstellung in einer Routing-Anwendung

Die restlichen fünf Teilnehmer entschieden sich für die Treppe, die neben dem Zielraum im Erdgeschoss liegt (vgl. Abb. 46.2). Grundsätzlich ähnelten sich die gezeichneten Routen dieser Teilnehmer. Die zwei Teilnehmer, die sich dafür entschieden, die Routen für die Anzeige innerhalb einer Anwendung anzupassen, zeichneten erneut annähernd rechtwinklige Strukturen.



Abbildung 46.2: Bearbeitungen der fünften Zeichenaufgabe  
Blau: Wege zur schnellsten Zielerreichung, rot: Wege für die Darstellung in einer Routing-Anwendung

*GraphHopper* löst die Aufgabe bei der Übergabe des Wegenetzwerks, das ohne weitere Funktionen erstellt worden war, nicht zufriedenstellend. So führt die Route über eine Treppe, für die sich keiner der Teilnehmer entschieden hatte, und durch einen weiteren Raum hindurch (vgl. Abb. 47.1).

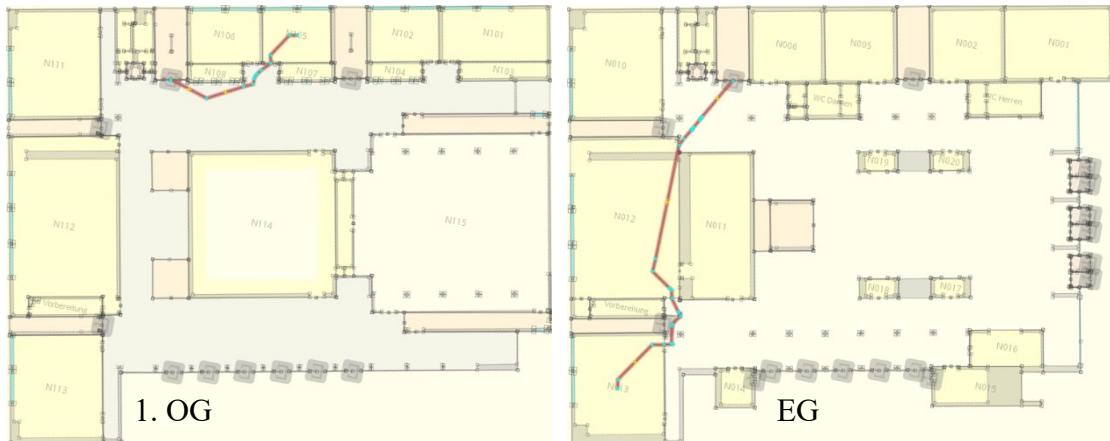


Abbildung 47.1: Durch GraphHopper berechnete Route für die fünfte Aufgabe  
Wegenetzwerk generiert ohne zusätzliche Funktionen

Bei den anderen Wegenetzwerken wurde die Treppe gewählt, für die sich die Hälfte der Teilnehmer entschieden hatte. Die Routen führen jedoch sehr dicht an den Wänden entlang und teilweise über zusätzliche Türen (vgl. Abb. 47.2).

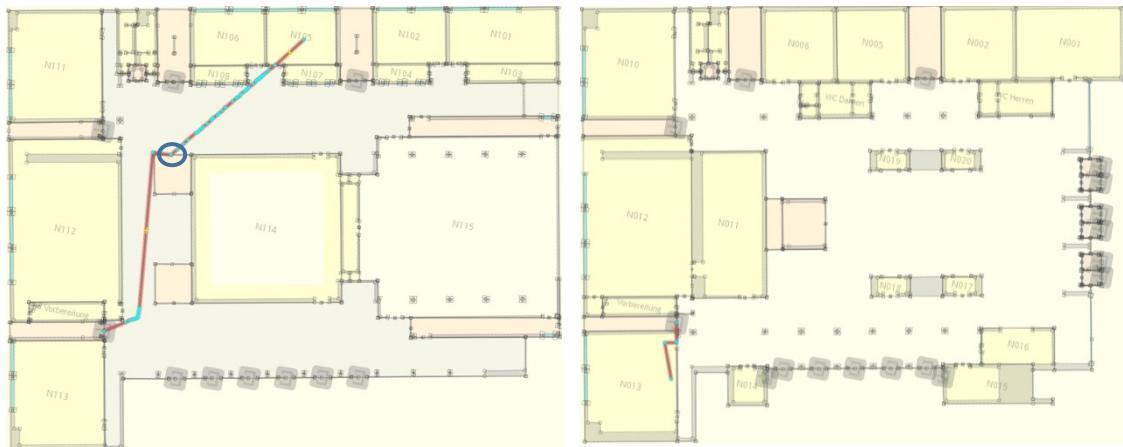


Abbildung 47.2: Durch GraphHopper berechnete Route für die fünfte Aufgabe  
Wegenetzwerk generiert mit simplify\_ways und door\_to\_door  
Blau umkreist: Tür (Verbindung zur Treppe), die nicht durchquert wird

### 7.1.5 Zusammenfassung und Schlussfolgerungen

Alle zehn Teilnehmer gaben in der anschließenden Befragung an, sich eine übersichtliche Darstellung der Routen in einer Indoor-Anwendung zu wünschen. Die Vorstellungen dazu, was als übersichtlich erachtet werden kann, gingen dabei teilweise auseinander. Die Zeichnungen haben jedoch gemein, dass keine Routen vorkommen, die sehr nah an den Wänden entlangführen. Damit wurde die Annahme bestätigt, Menschen für die Darstellung innerhalb einer Anwendung eher Wege bevorzugen, die auf Mittellinien verlaufen

In den ersten zwei Aufgaben ähneln die Ergebnisse der Teilnehmer sich. Auch unterscheiden sich die Routen aus dem ersten und zweiten Teil der Zeichenaufgabe jeweils nur unwesentlich. Die Unterschiede der gezeichneten Routen in der vierten und fünften Aufgabe sind hauptsächlich auf die jeweilige Wahl der Treppe zurückzuführen. Die größten Unterschiede in der Darstellung der Routen gibt es in der dritten Aufgabe, bei der eine große freie Fläche überquert werden musste. Dies spiegelt das grundsätzliche Problem wider, dass beim Indoor-Routing eine Lösung für das Überqueren von Flächen gefunden werden muss.

Sechs der Teilnehmer entschieden sich in mindestens vier der Aufgaben dafür, die Wege für die Darstellung innerhalb einer Anwendung anders zu zeichnen, als sie sie gehen würden. Eine weitere Person gab an, voreingenommen gewesen zu sein, da sie das Thema dieser Arbeit kannte. Aus diesem Grund hätte sie schon von Anfang an die Wege so gezeichnet, wie sie sie in einer Anwendung gerne angezeigt bekommen würde. Sieben von zehn Teilnehmern unterschieden also zwischen der Art, wie sich innerhalb von Gebäuden

fortbewegen, und wie sie Wege in einer Anwendung dargestellt bekommen möchten. Unter diesen sieben Teilnehmern befanden sich fünf der sechs Personen, die angegeben hatten, sich innerhalb von Gebäuden schlecht orientieren zu können. Obwohl die Darstellungen dieser Teilnehmer sich teilweise stark unterscheiden, haben sie gemeinsam, dass zugunsten von geraden Linien auf Kurven verzichtet wurde. Die Teilnehmer begründeten dies damit, sich eckige Formen besser merken zu können als Kurven. Außerdem seien die Richtungsanweisungen dadurch einfacher. Mehrere Personen gaben außerdem an, sich bei der Darstellung der Routen an Google Maps orientiert zu haben. Die drei Teilnehmer, deren Routen aus dem ersten Zeichenteil sich stärker von denen aus dem zweiten Teil unterschieden, erläuterten, dass sie die Wege so für übersichtlicher hielten und sich eventuelle Abkürzungen selbst suchen könnten.

Die restlichen drei Teilnehmer wünschten sich innerhalb einer Indoor-Routing-Anwendung eine Anzeige der Wege, wie sie sie auch gehen würden. Zwei dieser drei Teilnehmer gaben an, sich innerhalb von Gebäuden gut orientieren zu können. Es kann die Hypothese aufgestellt werden, dass Personen, die sich innerhalb von Gebäuden gut zurechtfinden können, auch leichter Wege nachvollziehen können, die in einer Anwendung dargestellt werden. Im Umkehrschluss bedeutete dies, dass Personen mit einem schlechteren Orientierungssinn sich eine schematischere Darstellung der Routen wünschen. Es wird davon ausgegangen, dass Personen mit einem guten Orientierungssinn auch mit einer schematischen Routendarstellung ihr Ziel leicht erreichen würden. Um diese Hypothesen zu überprüfen, müssten Befragungen und Tests in größerem Rahmen durchgeführt werden.

Beim Vergleich mit den von *GraphHopper* generierten Routen zeigte sich, dass die grundeliegenden Wegenetzwerke auf jeden Fall durch mindestens eine der Zusatzfunktionen überarbeitet werden müssen. Die generierten Routen ähneln grundsätzlich den Wegen, die die Teilnehmer im ersten Teil der Zeichenaufgaben zeichneten. Die eckigen Strukturen, die sich die Mehrzahl der Teilnehmer für die Darstellung innerhalb einer Anwendung wünscht, werden jedoch nicht wiedergegeben. Dies liegt vorrangig an dem dichten Wegenetzwerk, das durch das Tool generiert wird. Ein weiteres Problem ist, dass die von *GraphHopper* berechneten Wege oft über Türen führen, die nicht durchquert werden müssen.

## 7.2 Befragung der Teilnehmer

Die Befragung der Teilnehmer im zweiten Teil des Nutzertests zielte darauf ab, Genaues über die Erwartungen an eine Indoor-Routing-Anwendung herauszufinden. Die Teilnehmer sollten Situationen nennen, in denen sie sich vorstellen könnten, eine Indoor-Navigations-Anwendung zu nutzen. Des Weiteren sollten sie die Kriterien nennen, die für sie bei der Routengenerierung innerhalb eines Gebäudes wichtig wären. Abschließend wurde den Teilnehmern die Möglichkeit gegeben, weitere Wünsche und Erwartungen an eine Indoor-Anwendung zu äußern. Die Erkenntnisse aus der Befragung werden im Folgenden vorgestellt.

### 7.2.1 Mögliche Anwendungsszenarien

Alle Teilnehmer konnten sich vorstellen, eine Indoor-Routing-Anwendung vor allem in großen und unübersichtlichen Gebäuden zu verwenden. Die meisten Teilnehmer bezogen sich dabei direkt auf Universitätsgebäude. Als weitere Beispiele wurden die Verwendungen an Flughäfen, auf Messen, in Einkaufszentren und in Krankenhäusern genannt. Vier Teilnehmer schlugen zusätzlich eine Nutzung in öffentlichen Verwaltungsgebäuden wie Bürgeramt oder Rathaus vor. Drei Personen würden eine Indoor-Routing-Anwendung zum Kennenlernen von unbekannten Gebäuden nutzen, die sie häufiger besuchen möchten.

Acht Teilnehmer gaben an, eine Indoor-Routing-Anwendung sowohl im Voraus als auch direkt vor Ort verwenden zu wollen. Die Hälfte davon hielt dabei jedoch eine Verwendung im Voraus für wahrscheinlicher. Sie begründeten dies damit, vor einem wichtigen Termin, etwa einer Klausur oder einem Meeting, den Weg schon vorab wissen zu wollen, um ihr Ziel pünktlich zu erreichen. Zwei weitere Personen konnten sich eher vorstellen, vor Ort auf eine Indoor-Navigations-Anwendung zurückzugreifen. Sie schlossen aber die Verwendung im Voraus nicht aus. Zwei Personen waren beide Nutzungsszenarien gleich wichtig. Lediglich zwei Personen würden ausschließlich vor Ort die Hilfe einer Routing-Anwendung in Anspruch nehmen.

Drei Personen bevorzugten es, sich auf Beschilderungen, Lagepläne oder ortskundige Menschen vor Ort zu stützen. Auf eine Indoor-Navigations-Anwendung würden sie nur als letzte Maßnahme zurückgreifen. Eine weitere Person hielt es für sinnvoll, vor Ort Bildschirme aufzustellen, sodass ein Indoor-Routing ohne eigenes Gerät möglich ist.

### 7.2.2 Kriterien bei der Routenführung

Allen Teilnehmern war bei der Routenberechnung am wichtigsten, das Ziel so schnell wie möglich zu erreichen. Sie hielten jedoch auch die Möglichkeit, Routen nach eigenen Kriterien zu individualisieren, für sinnvoll. Fünf Personen schlugen die Option vor, barrierefreie Routen auswählen zu können. Es wurde dabei auf die Situation von Personen im Rollstuhl und solchen, die mit einem Koffer oder Wagen unterwegs sind, eingegangen. Die Teilnehmer regten an, dass ausgewählt werden könnte, ob Routen über Treppen, Rolltreppen oder Fahrstühle bevorzugt werden sollen. Vier Personen schlugen des Weiteren vor, in Gebäuden mit Innenhöfen und in Gebäudekomplexen je nach Wetterlage auswählen zu können, ob die errechnete Route durchs Freie führen darf.

Die meisten Teilnehmer hatten zunächst gezögert, Routen zu zeichnen, die durch Räume verlaufen<sup>13</sup>. So schlugen sie auch im Anschluss die Möglichkeit vor, auswählen zu können, ob andere Räume durchquert werden dürfen oder nicht. Dabei war den Teilnehmern wichtig, dass eine Anwendung das Durchqueren von Räumen nur vorschlägt, wenn sichergestellt ist, dass der Raum leer ist. Es wurde angeregt, grundsätzlich zwischen Räumen und Korridoren zu unterscheiden und letztere in der Routenplanung zu priorisieren.

Die Personen, die angaben, eine Indoor-Routing-Anwendung zum Kennenlernen von Gebäuden verwenden zu wollen, wünschten sich für diesen Anwendungsfall möglichst einfache Wege mit wenigen Anweisungen. Dafür würden sie auch Umwege in Kauf nehmen.

Sechs Personen wünschten sich die Möglichkeit, in ihrer Route auch Zwischenziele angeben zu können. Als Beispiele wurden neben einem weiteren Raum Umwege zur nächstgelegenen Toilette und der nächsten Möglichkeit zum Kaffeetrinken gegeben. Nach Auffassung der Teilnehmer sollten solche allgemeinen Ziele automatisch vorgeschlagen werden, ohne dass sie vom Benutzer eingegeben werden müssen.

---

<sup>13</sup> Wie schon erwähnt wurde den Teilnehmern mitgeteilt, dass sie davon ausgehen könnten, dass alle Räume leer sind und durchquert werden können. Aus diesem Grund entschieden sich trotzdem alle Teilnehmer in der ersten Aufgabe für einen Weg durch den Hörsaal.

### 7.2.3 Sonstige Erwartungen und Vorschläge

Am wichtigsten war den Teilnehmern eine zuverlässige Funktionsweise der Anwendung. Ein Teilnehmer gab das Beispiel, dass an einem Flughafen bei der Routenführung Sicherheitskontrollen, die nicht umgangen werden können, beachtet werden müssen. Dies spiegelt das von [WFH+17] angesprochene Problem wider, dass manche Wege nur in eine Richtung zulässig sind. Es ist ein weiteres Argument dafür, die automatisch generierten Wegenetzwerke noch einmal manuell zu überprüfen.

Ein Teilnehmer betonte, dass er von einer Routing-Anwendung erwarte, stets auf dem neusten Stand zu sein. So sollten bei Baumaßnahmen im Gebäude die Daten innerhalb der Anwendung parallel angepasst werden. Dies ist ein Argument für die Verwendung von *OSM*-Daten als Grundlage, da diese von jedem *OSM*-Benutzer angepasst werden können. So kann sehr schnell auf Änderungen reagiert werden.

Die Teilnehmer wünschten sich eine Anwendung, die intuitiv zu bedienen ist. In Bezug auf mehrstöckige Gebäude erwarteten sie, stets die aktuelle Etage angezeigt zu bekommen. Bei einem Etagenwechsel sollten außerdem die Ausgangs- und Zieletage klar erkennbar sein. Treppen, Fahrstühle und Türen sollten durch Symbole hervorgehoben werden. Dies sollte bei der Integrierung der Routing-Methode in das Framework zur Darstellung von Gebäudedaten beachtet werden.

Mehrere Teilnehmer gaben an, dass sie als Ziel nicht eine bestimmte Position innerhalb eines Raumes angeben würden. Stattdessen würde es ihnen genügen, zur Tür des Zielraums geführt zu werden. Eine Person regte an, eine Indoor-Routing-Anwendung zu entwickeln, die über eine Spracheingabe bedient werden kann. Eine weitere Person wünschte sich die Möglichkeit eines Wechsels zwischen einer 2D- und 3D-Ansicht des Gebäudes.

Die Mehrzahl der Teilnehmer würden eine Webanwendung bevorzugen. Die Installation einer Desktopanwendung oder einer mobilen Applikation konnten sie sich nur für Gebäude vorstellen, die sie öfter besuchen möchten. Zwei Teilnehmer regten an, innerhalb der Gebäude QR-Codes anzubringen, um einen schnellen Zugriff auf eine Indoor-Routing-Webanwendung zu ermöglichen.

## 8 Fazit

In großen und unübersichtlichen Gebäuden verlieren viele Menschen schnell die Orientierung. Gerade für Menschen, die auf barrierefreie Wege angewiesen sind, kann dies eine große Herausforderung darstellen. Es bietet sich daher an, für solche Gebäude eine Indoor-Routing-Anwendung zur Verfügung zu stellen. Aufbauend auf einem zuvor abgeschlossenen Forschungsseminar beschäftigte sich die vorliegende Abschlussarbeit damit, eine Methode für das Indoor-Routing auf Basis von *OpenStreetMap*-Daten zu entwickeln. Die zu Anfang gestellten Ziele wurden dabei wie folgt erreicht:

1. Das *Simple-Indoor-Tagging*-Format wurde ergänzt, um ein einheitliches Schema für die Beschreibung von Gebäudedaten im *OSM*-Format zu schaffen. Insbesondere werden nun zusammenhängende Treppenteile und Fahrstuhlhaltepunkte, die mehrere Etagen verbinden, in Relationen gespeichert.
2. Das Tool zur automatischen Generierung von Wegenetzwerken wurde überarbeitet:
  - a. Beim Algorithmus für die Vereinfachung von Wegen innerhalb eines Raums muss kein Schwellenwert mehr übergeben werden.
  - b. Die generierten Wegenetzwerke enthalten nun auch Verbindungen über mehrere Etagen hinweg.
  - c. Die Funktion zur Generierung der Wege auf Basis des *Straight Skeletons* berücksichtigt nun Barrieren innerhalb von Räumen und Korridoren.
  - d. Optional können zusätzlich direkte Verbindungen zwischen Türen innerhalb eines Raumes oder Korridors als Wege gespeichert werden.
3. Die quelloffene Software *GraphHopper*, die bereits für das Routing im Außenbereich eingesetzt wird, wurde dem Routing innerhalb von Gebäuden angepasst. Mit Hilfe von *Flag Encodern* kann die Routenführung individualisiert werden. In Hinblick auf körperlich Beeinträchtigte wurde die Möglichkeit implementiert, Routen über Treppen oder Fahrstühle auszuschließen.
4. Für die Auswertung der von *GraphHopper* berechneten Routen auf Basis der automatisch generierten Wegenetzwerke wurde ein Nutzertest mit zehn Teilnehmern durchgeführt. Die von *GraphHopper* erstellten Routen ähnelten grundsätzlich den Routen, für die die Teilnehmer sich entschieden. Die besten Ergebnisse wurden mit den Wegenetzwerken erzielt, die mit dem Vereinfachungsalgorithmus

überarbeitet und/oder um direkte Verbindungen zwischen Türen innerhalb eines Raumes ergänzt worden waren.

Die Teilnehmer am Nutzertest bekundeten ein grundsätzliches Interesse daran, eine Indoor-Routing-Anwendung zu verwenden. Als Anwendungsbeispiele nannten sie ähnliche wie die zuvor recherchierten. Den Nutzen für körperlich Beeinträchtigte hoben sie ebenfalls hervor. Beim Indoor-Routing auf Basis von *OSM*-Daten handelt es sich demnach um eine vielversprechende Methode, um Menschen das Zurechtfinden in unbekannten komplexen Gebäuden zu erleichtern. Nach bestem Wissen wurde in dieser Arbeit die erste Methode entwickelt, bei der die zugrundeliegenden Daten im *Simple-Indoor-Tagging*-Schema beschrieben wurden. Die vom Tool automatisch generierten Wegenetzwerke sollten zur Sicherheit manuell überprüft werden. Nichtsdestotrotz dürfte dieser Ansatz deutlich weniger Zeit in Anspruch nehmen, als das gesamte Wegenetzwerk innerhalb eines Gebäudes von Hand einzuziehen.

## 9 Ausblick

Bei der Beschreibung von Gebäuden im *OSM*-Format und der automatischen Generierung von Wegenetzwerken wurde in Bezug auf Verbindungen über mehrere Stockwerke hinweg nur auf Treppen und Fahrstühle eingegangen. Der Grund dafür liegt darin, dass es innerhalb des Neuen Hörsaalgebäudes der TU Chemnitz, das als Testgelände diente, keine anderen Arten von Verbindungen zwischen Etagen gibt. In Zukunft sollten jedoch auch andere Verbindungen, wie Rolltreppen oder Rampen, berücksichtigt werden. Der Algorithmus für die Generierung von Wegen über Treppen sollte zudem angepasst werden, sobald sich die *OSM*-Community auf eine Lösung für die genauere Kartographierung von Treppen im *OSM*-Format geeinigt hat.

Das Tool zur automatischen Generierung von Wegenetzwerken sollte so erweitert werden, dass Wege ausgeschlossen werden, die sehr dicht an den Wänden entlangführen. Da teilweise sehr dichte Netzwerke gebildet werden, sollte außerdem eine Funktion entwickelt werden, die dicht beieinander liegende Punkte zu einem einzigen Punkt vereinigt.

Es zeigte sich, dass die verwendete *Straight-Skeleton*-Implementierung *polyskel* bei vorhandenen Barrieren innerhalb des Raumes nicht immer zuverlässig funktioniert. Um etwas bessere Ergebnisse zu erzielen, könnten nahe beieinander liegende Barrieren zu einer Barriere zusammengefasst werden. Nach Möglichkeit könnte *polyskel* auch durch eine bessere Implementierung ersetzt werden. Je nach Bedarf sollte die als Übergangslösung geschriebene Funktion *add\_supplementary\_ways* dann wieder entfernt werden.

Generell sollte bei der Generierung von Wegen in Zukunft darauf eingegangen werden, ob ein Raum oder Korridor öffentlich zugänglich ist oder nicht. Des Weiteren wäre es sinnvoll, Fluchtwege als solche zu kennzeichnen. Auch eventuelle Öffnungszeiten sollten berücksichtigt werden. Sollten in Zukunft Gebäude mit Rolltreppen betrachtet werden, müsste zusätzlich auch auf die Richtung, in die diese führen, eingegangen werden. Dies gilt auch für Wege durch Drehkreuze oder andere Strukturen, die sich nur in eine Richtung durchqueren lassen.

*GraphHopper* berechnet häufig Routen, die über Türen führen, die nicht durchquert werden müssen. Aus diesem Grund sollten die *Flag Encoder* so angepasst werden, dass sie Routen über Eingänge, falls möglich, ausschließen. Dafür müssten die Knoten, die Ein-

gänge darstellen, bereits bei der automatischen Generierung der Wegenetzwerke als solche gekennzeichnet werden. Der Vorschlag der Teilnehmer am Nutzertest, beim Routing zwischen Räumen und Korridoren zu unterscheiden, erscheint ebenso sinnvoll. Dafür müssten die generierten Wege unterschiedliche *Tags* erhalten, die diese als Wege durch Korridore oder Räume kennzeichnen. Es sollte kein Problem darstellen, dies bei der Generierung der Wegenetzwerke zu berücksichtigen. Je nach Bedarf sollte auch auf weitere Kriterien eingegangen werden.

Bei mehrstöckigen Gebäuden nimmt *GraphHopper* für Knoten des Navigationsgraphen, die auf unterschiedlichen Etagen direkt übereinander liegen, eine Entfernung von 0 Metern an. Dadurch können Routen gebildet werden, die mehr Etagenwechsel als nötig enthalten. Außerdem wird die Gesamtlänge der Route deswegen nicht korrekt wiedergegeben. *GraphHopper* sollte deshalb zukünftig die Entfernung zwischen zwei Knoten, die sich auf unterschiedlichen Etagen befinden, anders berechnen.

Die Positionsbestimmung innerhalb von Gebäuden war kein Bestandteil dieser Arbeit. Trotzdem spielt sie eine wichtige Rolle beim Routing. So sollte stets gewährleistet sein, dass ein Benutzer einer Indoor-Routing-Anwendung auf der richtigen Etage verortet wird. Die Funktionalität von *GraphHopper* sollte so ausgebaut werden, dass als Ziele Raumnummern oder wichtige Punkte innerhalb des Gebäudes anstelle von Koordinaten eingegeben werden können. Ziele sollten auch direkt vorgeschlagen werden. Die Wegbeschreibungen, die *GraphHopper* generiert, sollten an die Begebenheiten innerhalb von Gebäuden angepasst werden.

Der Vorschlag zweier Teilnehmer, für einen schnellen Zugriff auf eine Indoor-Routing-Webanwendung innerhalb des Gebäudes QR-Codes anzubringen, wird als vielversprechend angesehen und dürfte sich leicht umsetzen lassen.

Um genauere Annahmen darüber treffen zu können, welche Darstellungsweisen von Routen innerhalb von Gebäuden als übersichtlich erachtet werden, sollte eine Studie in größerem Rahmen durchgeführt werden.

## Literaturverzeichnis

- [AFA+14] AMAT, Guillermo; FERNANDEZ, Javier; ARRANZ, Alvaro; RAMOS, Angel: Using Open Street Maps data and tools for indoor mapping in a Smart City scenario. In: HUERTA GUIJARRO, Joaquín; SCHADE, Sven; GRANELL CANUT, Carlos (Hrsg.): *Connecting a Digital Europe through Location and Place. Proceedings of the AGILE'2014 International Conference on Geographic Information Science*. Castellón, 2014.
- [Aus18] AUSCHRA, Bettina: *Automatische Generierung von Navigationsgraphen auf Basis von OpenStreetMap-Innenraumkarten*. Technische Universität Chemnitz: Forschungsseminar, 2018.
- [BFB+06] BEALE, Linda; FIELD, Kenneth; BRIGGS, David; PICTON, Phil; MATTHEWS, Hugh: Mapping for wheelchair users: Route navigation in urban spaces. In: *The Cartographic Journal* 43 (2006), Nr. 1, S. 68-81.
- [BG04] BÜCHEL, Daniela; GILLIÉRON, Pierre-Yves: Navigation pédestre à l'intérieur des bâtiments. *Géomatique Suisse* 102 (2004), Nr 11, S. 664-668.
- [DMV+18] DELNEVO, Giovanni; MONTI, Lorenzo; VIGNOLA, Francesco; SALOMONI, Paola; MIRRI, Silvia: AlmaWhere: A prototype of accessible indoor wayfinding and navigation system. In: *15th Annual IEE Consumer Communications & Networking Conference*. Las Vegas: IEEE, 2018, S. 1-6.
- [FAB+13] FALLAH, Navid; APOSTOLOPOULOS, Ilias; BEKRIS, Kostas; FOLMER, Eelke: Indoor human navigation systems: A survey. *Interacting with Computers* 25 (2013), Nr 1, S. 21-33.
- [FHG17] FELLNER, Irene; HUANG, Haosheng; GARTNER, Georg: "Turn Left after the WC, and Use the Lift to Go to the 2nd Floor"—Generation of Landmark-Based Route Instructions for Indoor Navigation. In: *ISPRS International Journal of Geo-Information* 6 (2017), Nr. 6.
- [FMS+14] FRUTH, Robert; MOEWS, Henry; SILVA-RIBEAUX, David; TSCHERNACK, Tom; WALL, Norbert; WOLFIEN, Patrick: Campus-Navigations-Applikation für Studierende mit Beeinträchtigungen. In: Plödereder, E., Grunske, L., Schneider, (Hrsg.): *Informatik 2014*. Bonn: Gesellschaft für Informatik e.V., S. 2401-2412.
- [GZ12] Goetz, Marcus; Zipf, Alexander: Mapping the Indoor World: Towards Crowdsourcing Geographic Information About Indoor Spaces. In: *GIM international*, 26 (2012), Nr. 3, S. 30-34.
- [HA01] HORMANN, Kai; AGATHOS, Alexander: The point in polygon problem for arbitrary polygons." *Computational Geometry* 20 (2001), Nr. 3, S. 131-144.
- [HS08] HAUNERT, Jan-Henrik; SESTER, Monika: Area collapse and road centerlines based on straight skeletons. *GeoInformatica* 12 (2008), Nr. 2 , S. 169-191.

- [KDO+10] KAMMOUN, Slim; DRAMAS, Florian; ORIOLAAND, Bernard; JOUFFRAIS, Christophe: Route Selection Algorithm for Blind Pedestrian. In: *International Conference on Control, Automation and Systems*. Gyeonggi-do: IEEE, 2010, S. 2223-2228.
- [LLC+09] LERTLAKKHANAKUL, Jumphon; LI, Yongzhi; CHOI, Jinwon; BU, Soyoungh: GongPath: Development of BIM based indoor pedestrian navigation system. In: *Fifth International Joint Conference on INC, IMS and IDC*. Seoul: IEEE, 2009, S. 382-388.
- [LSA08] LYARDET, Fernando; SZETO, Diego Wong; AITENBICHLER, Erwin: Context-Aware Indoor Navigation. In: AARTS, Emile; CROWLEY, James L.; DE RUYTER, Boris: *European Conference, AmI 2008, Nuremberg, Germany, November 19-22, 2008. Proceedings*. Berlin: Springer, 2008, S. 290-307.
- [LZ11] LIU, L.; ZLATANOVA, S.: A "door-to-door" path-finding approach for indoor navigation. In: *Proceedings Gi4DM 2011: GeoInformation for Disaster Management*. Antalya: 2011.
- [Nei15] NEIS, Pascal: Measuring the Reliability of Wheelchair User Route Planning based on Volunteered Geographic Information. In: *Transactions in GIS* 19 (2015), Nr 2, S. 188–201
- [OSMa] OPENSTREETMAP: *History of OpenStreetMap*. [https://wiki.openstreetmap.org/wiki/History\\_of\\_OpenStreetMap](https://wiki.openstreetmap.org/wiki/History_of_OpenStreetMap). Aufgerufen am 21.07.2018
- [OSMb] OPENSTREETMAP: *OpenStreetMap Statistics*. [https://www.openstreetmap.org/stats/data\\_stats.html](https://www.openstreetmap.org/stats/data_stats.html). Aufgerufen am 14.11.2018.
- [OSMc] OPENSTREETMAP: *Willkommen bei OpenStreetMap*. [https://wiki.openstreetmap.org/wiki/Willkommen\\_bei\\_OpenStreetMap](https://wiki.openstreetmap.org/wiki/Willkommen_bei_OpenStreetMap). Aufgerufen am 21.07.2018.
- [OSMd] OPENSTREETMAP: *Elements*. <https://wiki.openstreetmap.org/wiki/Elements>. Aufgerufen am 21.07.2018.
- [OSMe] OPENSTREETMAP: *Tags*. <https://wiki.openstreetmap.org/wiki/Tags>. Aufgerufen am 21.07.2018.
- [OSMf] OPENSTREETMAP: *Relation*. <https://wiki.openstreetmap.org/wiki/Relation>. Aufgerufen am 21.07.2018.
- [OSMg] OPENSTREETMAP: *Node*. <https://wiki.openstreetmap.org/wiki/Node>. Aufgerufen am 21.07.2018.
- [OSMh] OPENSTREETMAP: *Way*. <https://wiki.openstreetmap.org/wiki/Way>. Aufgerufen am 21.07.2018.
- [OSMi] OPENSTREETMAP: *Simple Indoor Tagging*. [https://wiki.openstreetmap.org/wiki/Simple\\_Indoor\\_Tagging](https://wiki.openstreetmap.org/wiki/Simple_Indoor_Tagging). Aufgerufen am 07.08.2018.
- [OSMj] OpenStreetMap: *IndoorOSM*. [https://wiki.openstreetmap.org/wiki/Proposed\\_features/IndoorOSM](https://wiki.openstreetmap.org/wiki/Proposed_features/IndoorOSM). Aufgerufen am 07.08.2018.
- [OSMk] Routing. In: *OpenStreetMap Wiki*. <https://wiki.openstreetmap.org/wiki/Routing>. Aufgerufen am 08.08.2018.

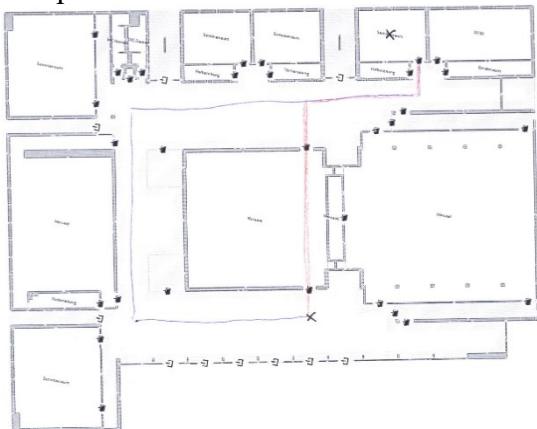
- [OSM1] Key:highway. In: *OpenStreetMap Wiki*. <https://wiki.openstreetmap.org/wiki/Key:highway>. Aufgerufen am 21.07.2018.
- [OSMm] Tag:highway=footway. In: *OpenStreetMap Wiki*. <https://wiki.openstreetmap.org/wiki/Tag:highway=footway>. Aufgerufen am 21.07.2018.
- [SSO08] STOFFEL, Edgar-Philipp; SCHODER, Korbinian; OHLBACH, Hans Jürgen: Applying hierarchical graphs to pedestrian indoor navigation. In: *16th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems*. Irvine: ACM-GIS, 2008.
- [SSS74] SUTHERLAND, Ivan E.; SPROULL, Robert F.; SCHUMACKER, Robert A.: A characterization of ten hidden-surface algorithms. In: *ACM Computing Surveys (CSUR)* 6 (1974), Nr. 1, S. 1-55.
- [Sun] SUNDAY, Dan: Inclusion of a Point in a Polygon. [http://geomalgorithms.com/a03\\_inclusion.html](http://geomalgorithms.com/a03_inclusion.html). Aufgerufen am 25.08.2018
- [TAK+05] TSETSOS, Vassileios; ANAGNOSTOPOULOS, Christos; KIKIRAS, Panayotis; HASIOTIS, P; HADJIEFTHYMIADES, Stathes: A human-centered semantic navigation system for indoor environments. In: *International Conference on Pervasive Services*. Santorini: IEEE, 2005.
- [WFH+17] WAGNER, Stephan; FET, Ngewi; HANDTE, Marcus; MARRÓN, Pedro José: An Approach for Hybrid Indoor/Outdoor Navigation. In: *13th 2017 International Conference on Intelligent Environments*. Seoul: IEEE, 2017, S. 36-43.
- [WY15] WORBOYS, Michael, YANG, Liping: Generation of navigation graphs for indoor space. In: *International Journal of Geographical Information Science* 10 (2015), Nr. 29, S. 1737-1756.
- [ZMR+16] ZIPF, Alexander; MOBASHERI, Amin; ROUSELL, Adam; HAHMANN, Stefan: Crowdsourcing for individual needs—The case of routing and navigation for mobility-impaired persons. In: CAPINERI, Cristina; HAKLAY, Muki; HUANG, Haosheng (Hrsg.): *European Handbook of Crowdsourced Geographic Information*. London: Ubiquity Press, 2016, S. 325-337.

## Anhang

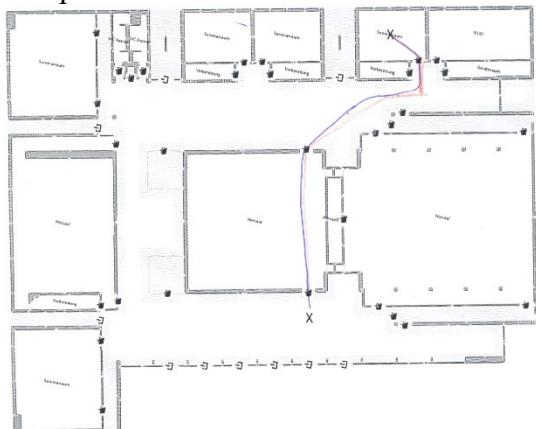
## Bearbeitungen der Zeichenaufgaben

### Zeichenaufgabe 1

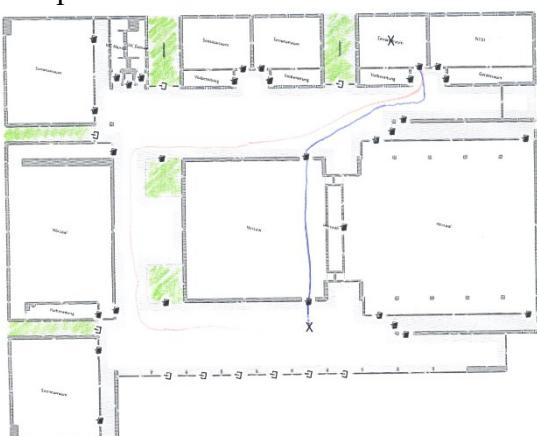
Testperson 1:



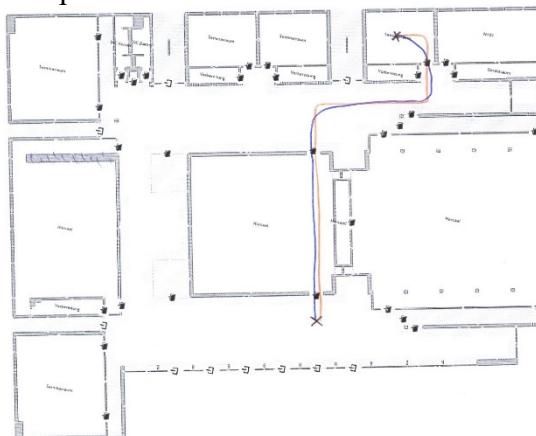
Testperson 2:



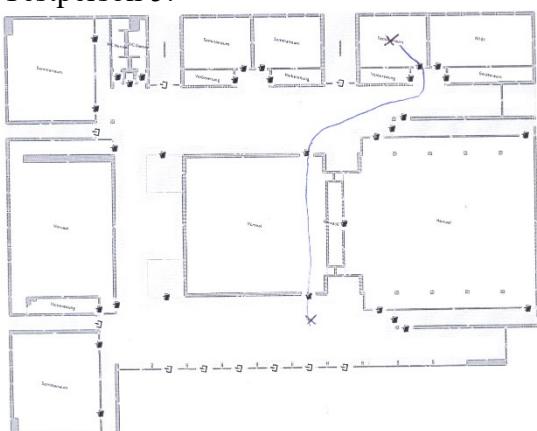
Testperson 3:



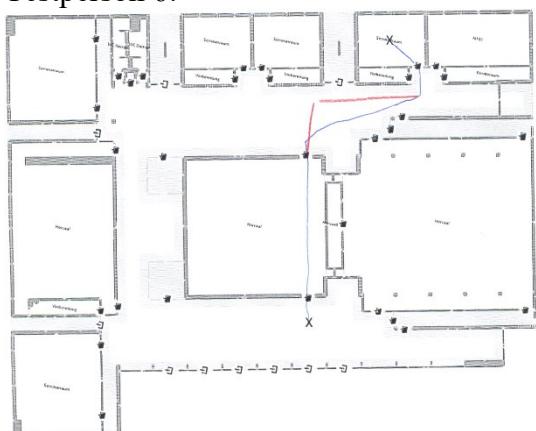
Testperson 4:



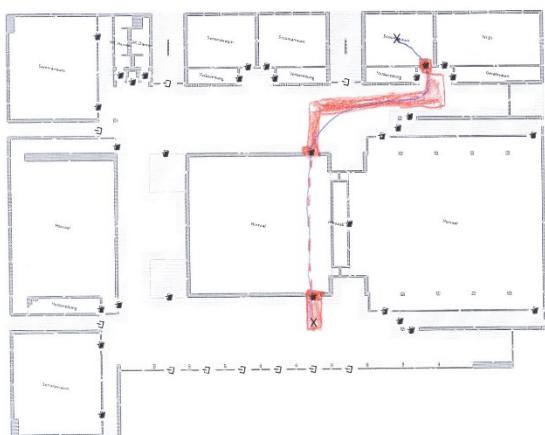
Testperson 5:



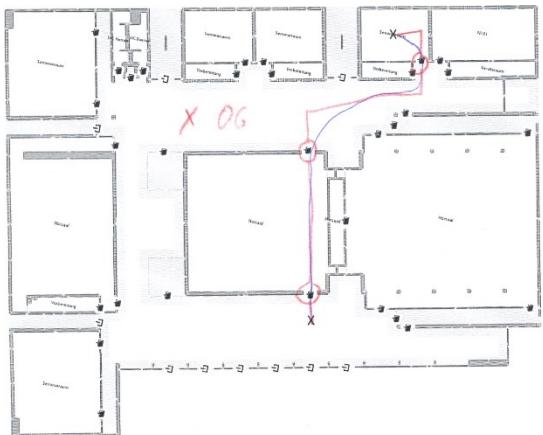
Testperson 6:



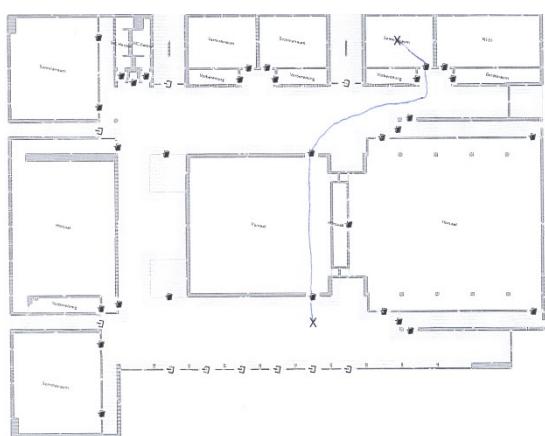
Testperson 7:



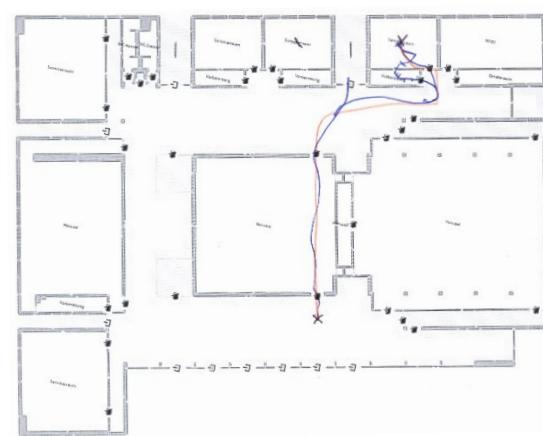
Testperson 8:



Testperson 9:

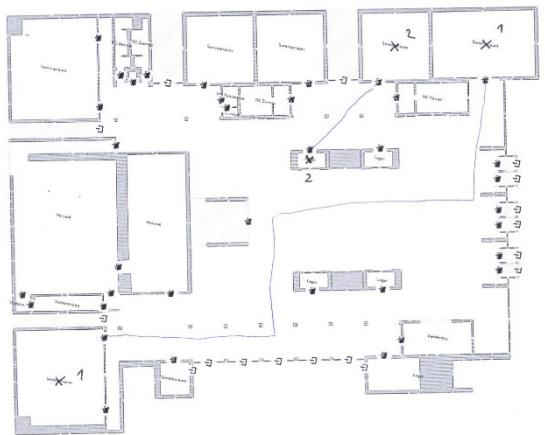


Testperson 10:

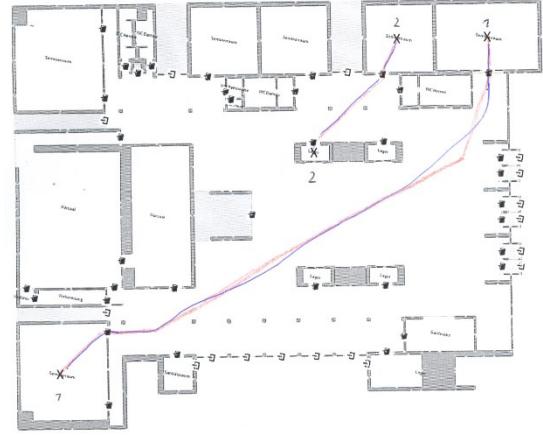


## Zeichenaufgaben 2 und 3

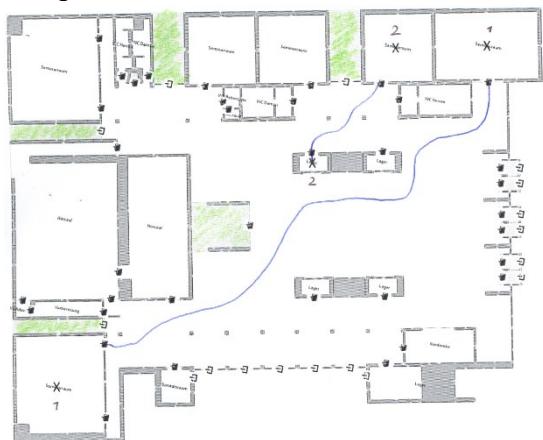
Testperson 1:



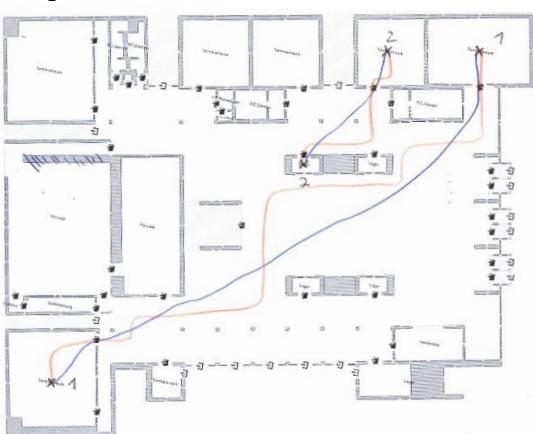
Testperson 2:



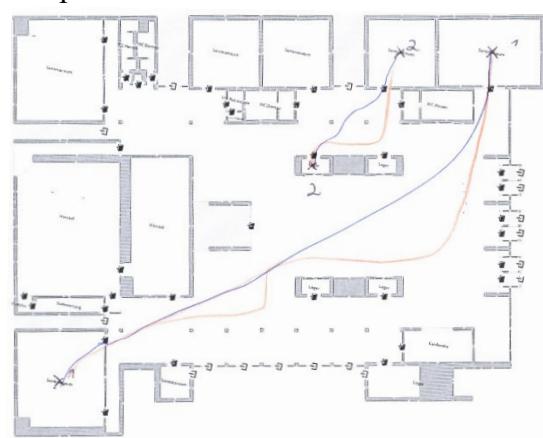
Testperson 3:



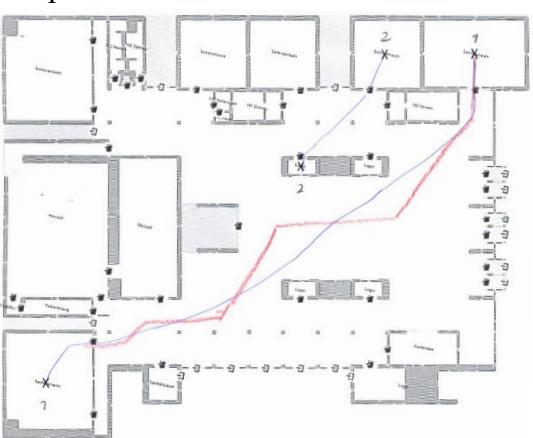
Testperson 4:



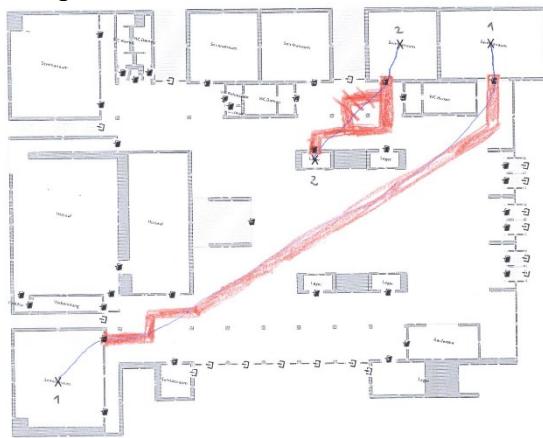
Testperson 5:



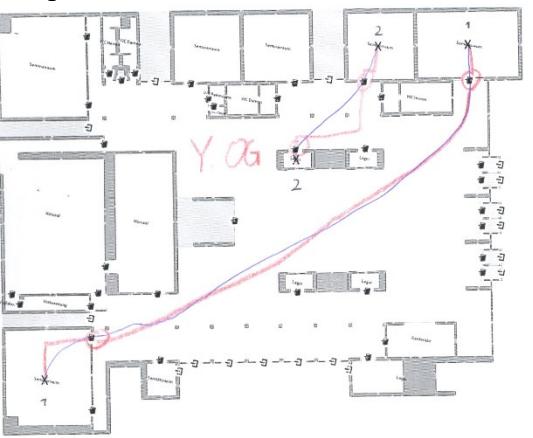
Testperson 6:



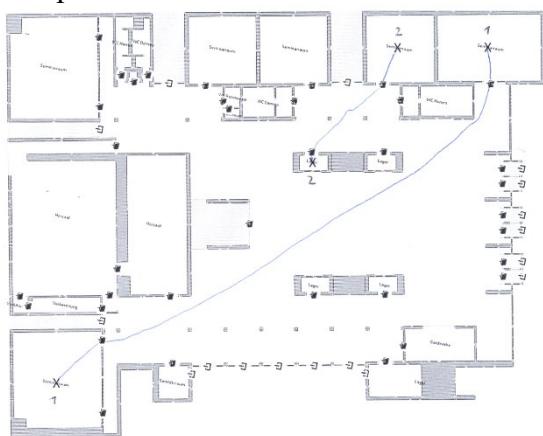
Testperson 7:



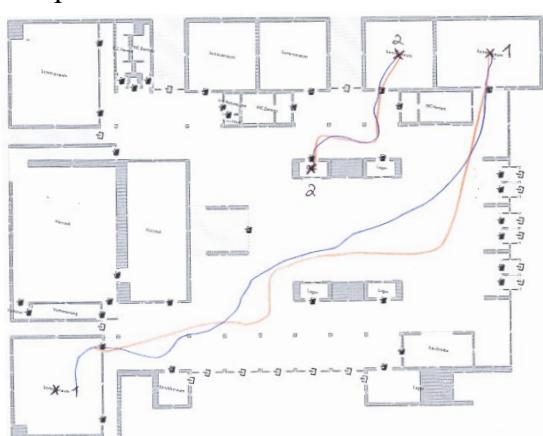
Testperson 8:



Testperson 9:

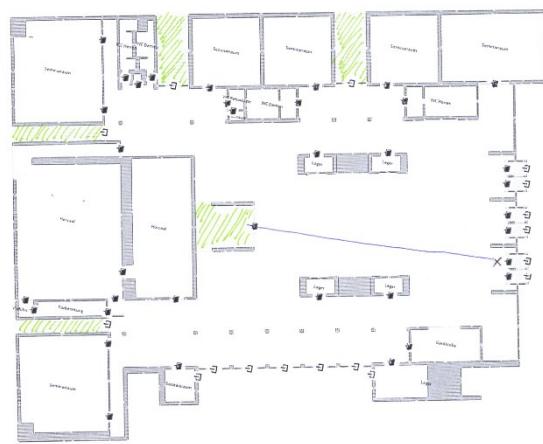
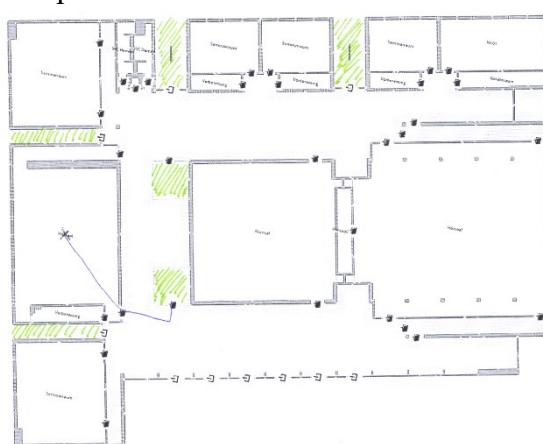


Testperson 10:

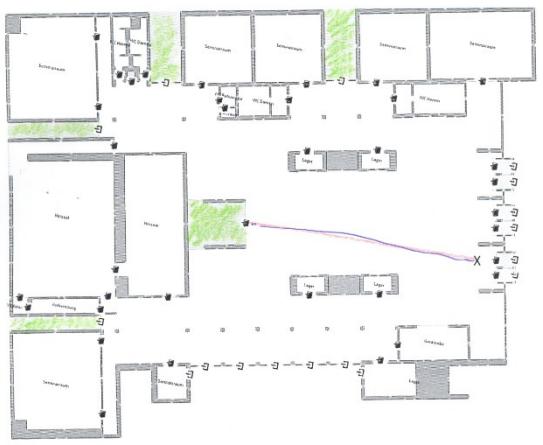
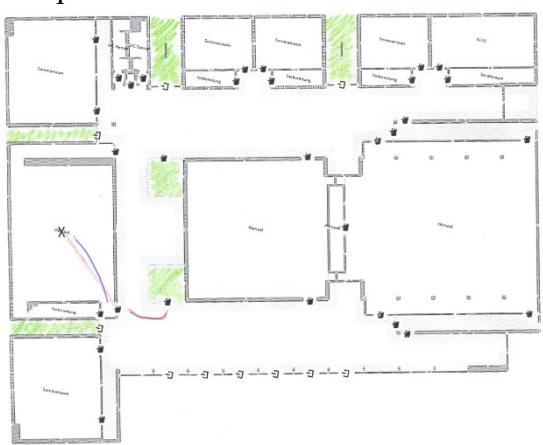


#### Zeichenaufgabe 4

Testperson 1:



Testperson 2:



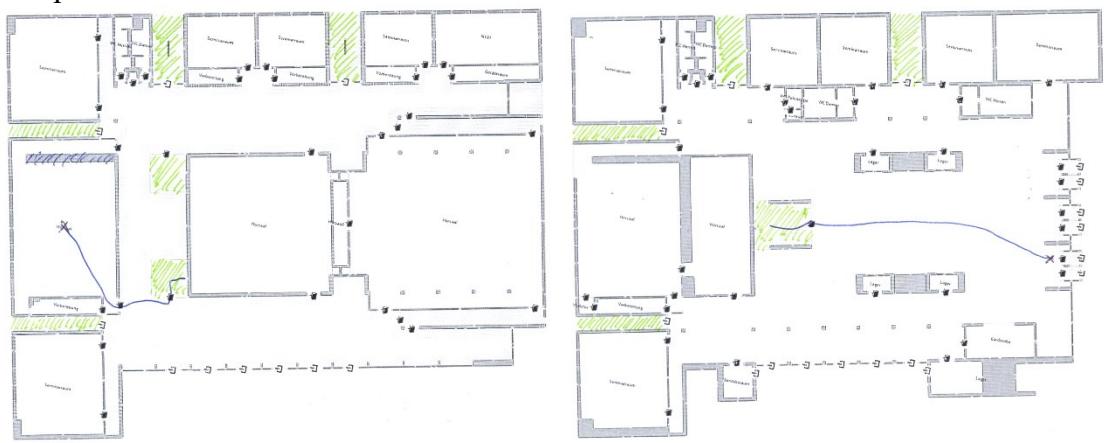
Testperson 3:



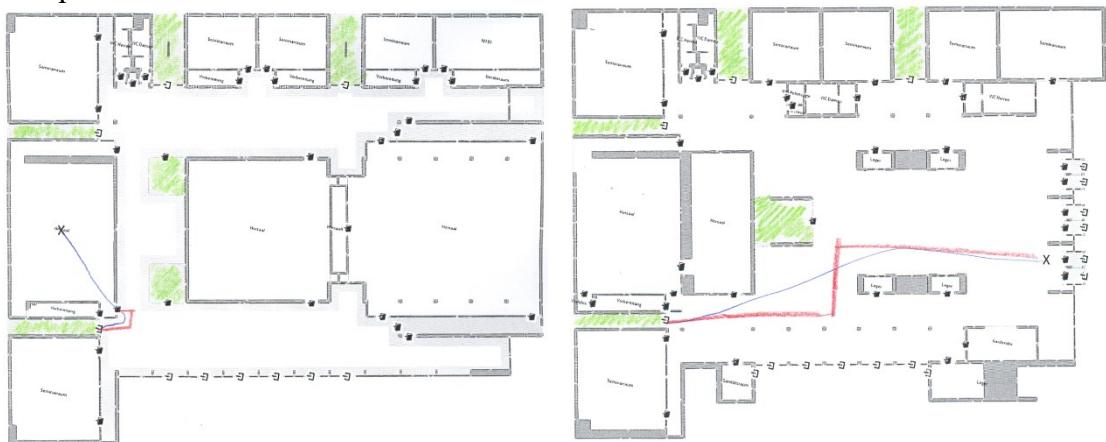
Testperson 4:



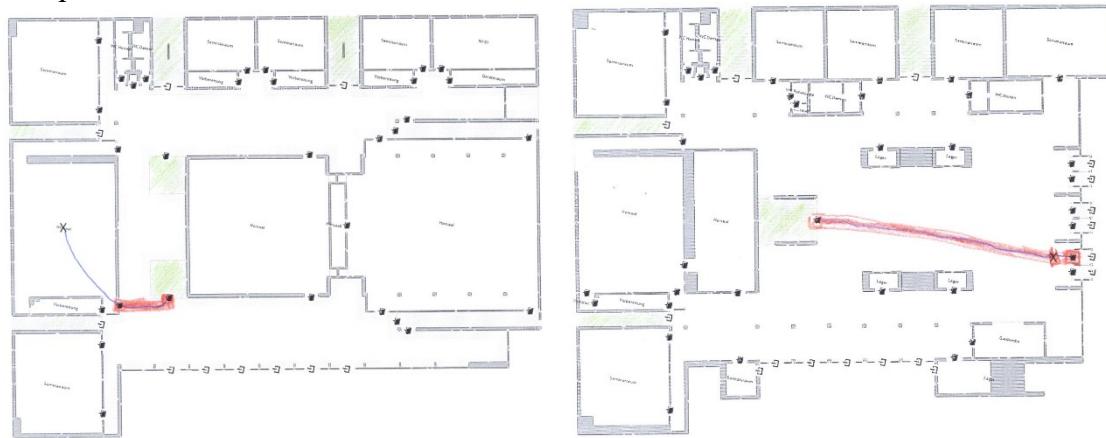
Testperson 5:



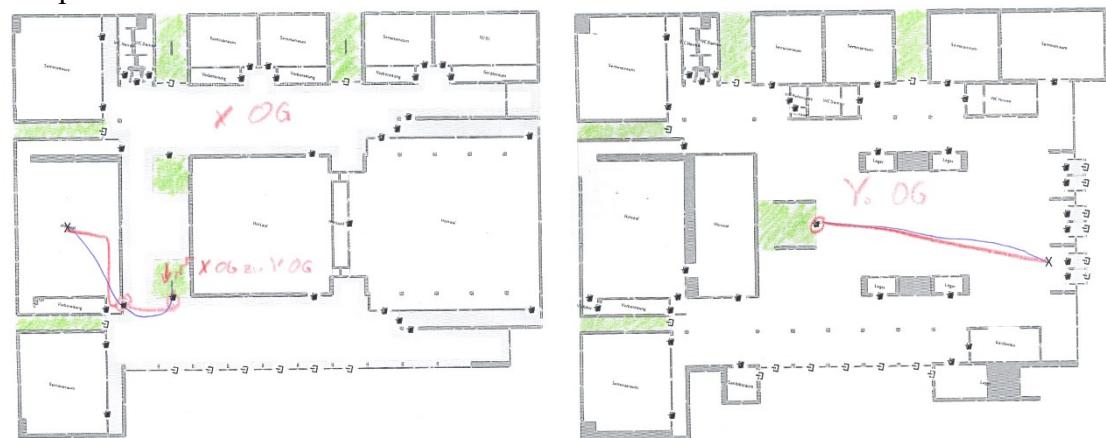
Testperson 6:



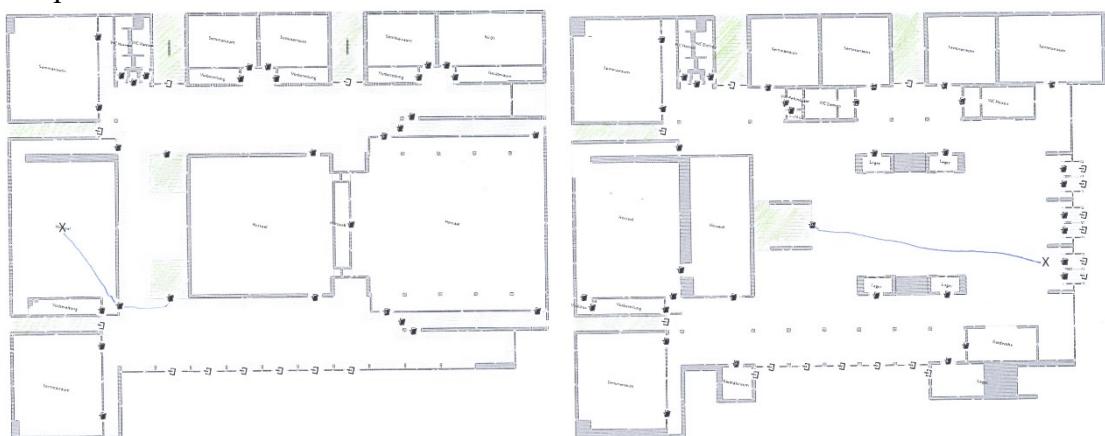
Testperson 7:



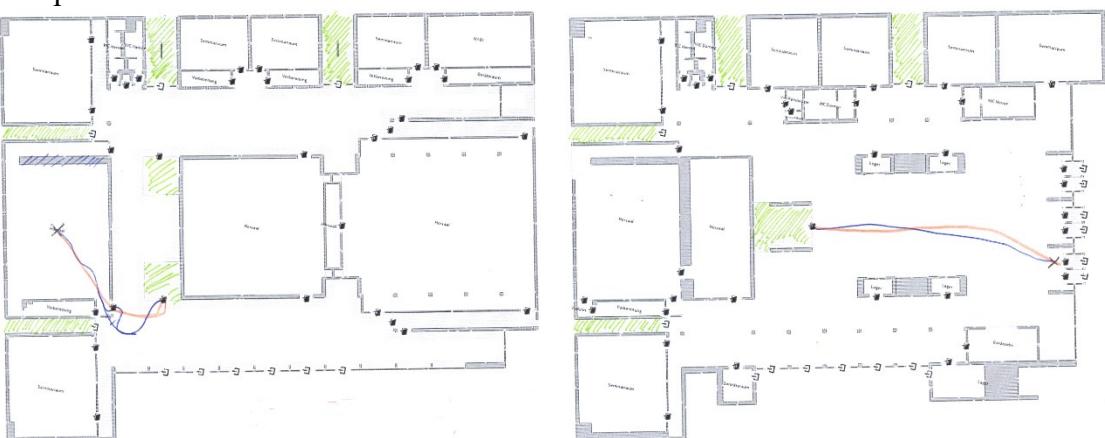
Testperson 8:



Testperson 9

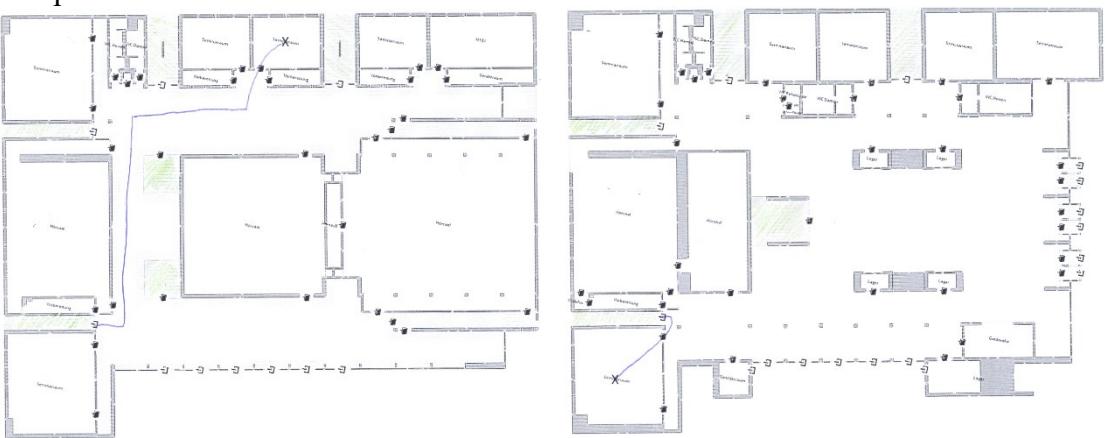


Testperson 10:



### Zeichenaufgabe 5

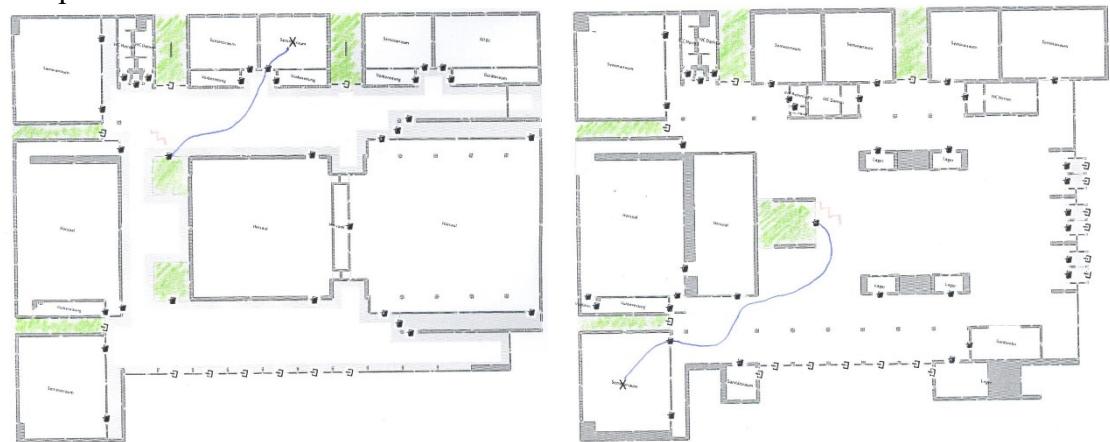
Testperson 1:



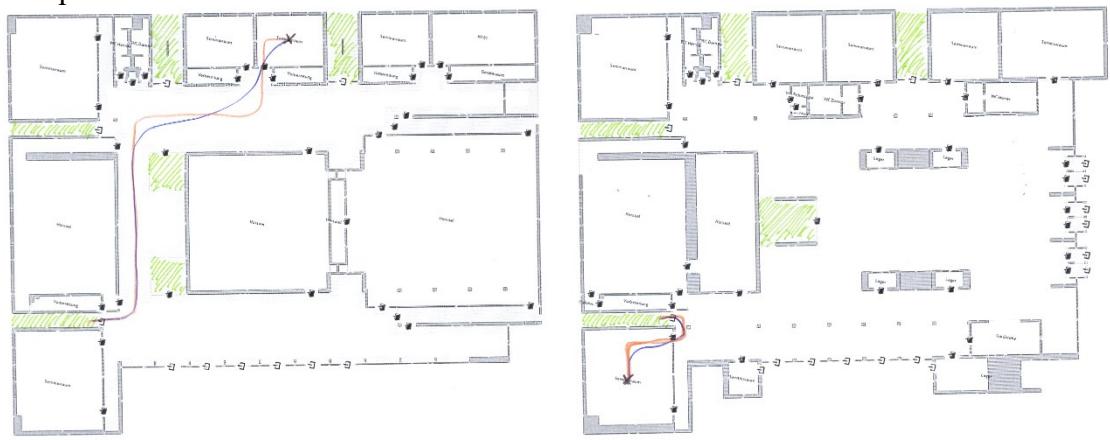
Testperson 2:



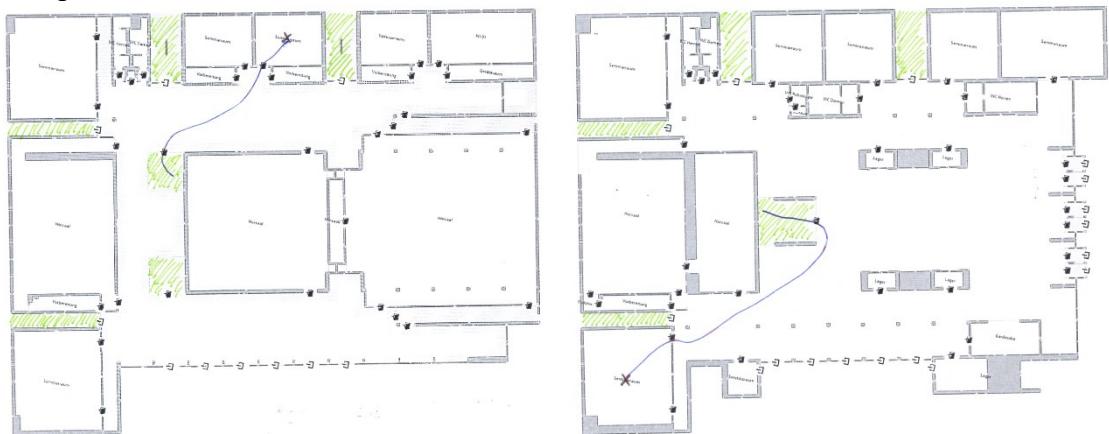
Testperson 3:



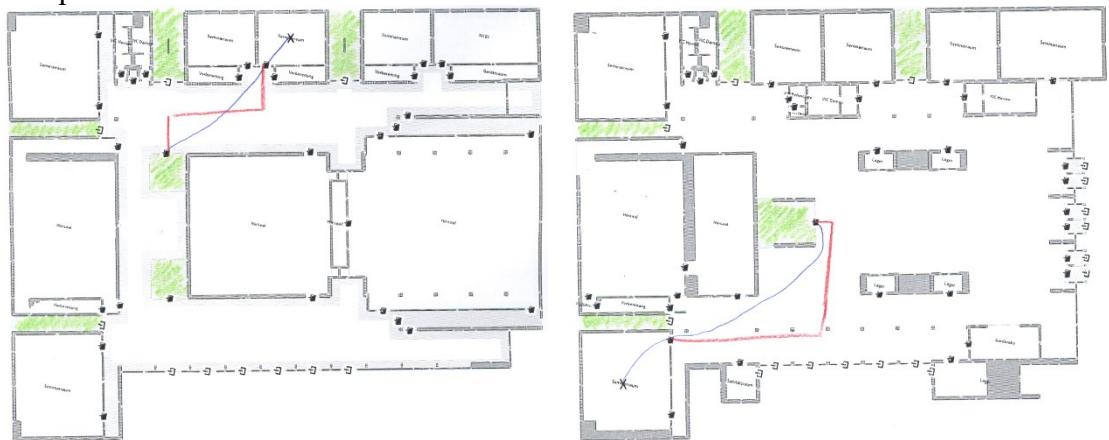
Testperson 4:



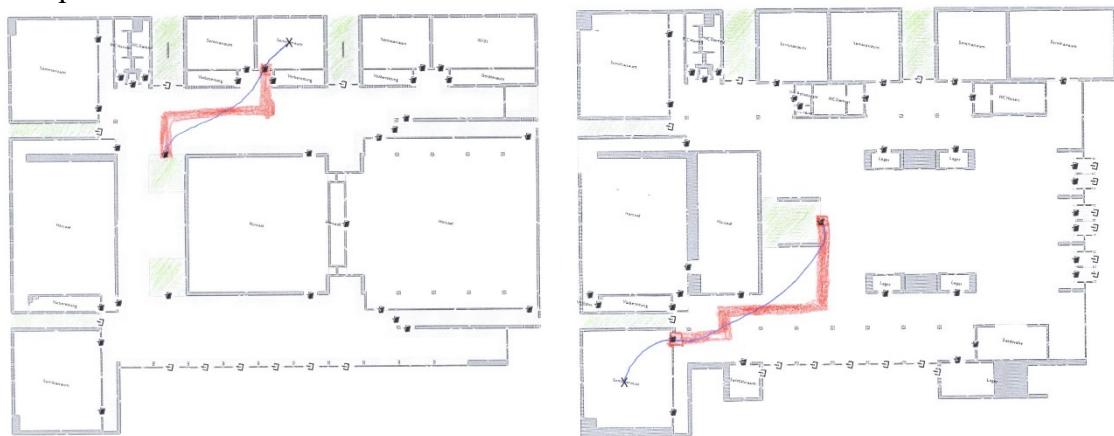
Testperson 5:



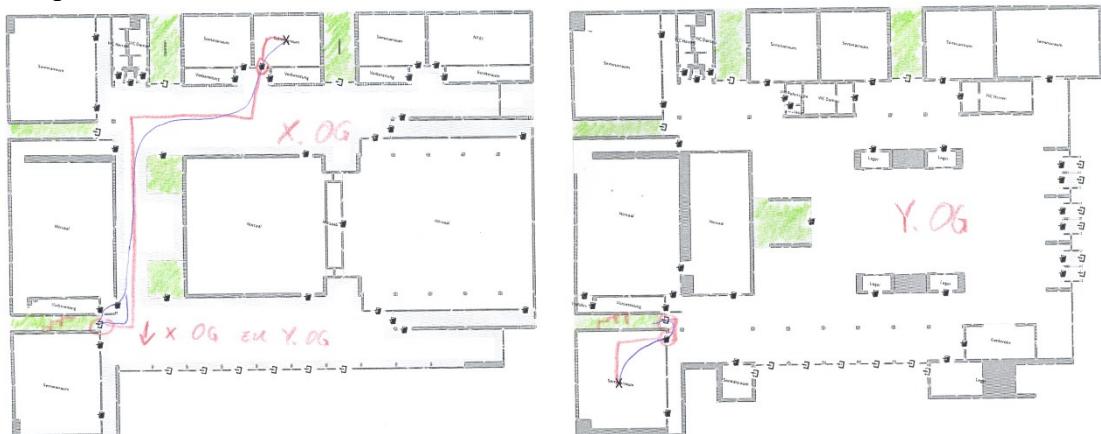
Testperson 6:



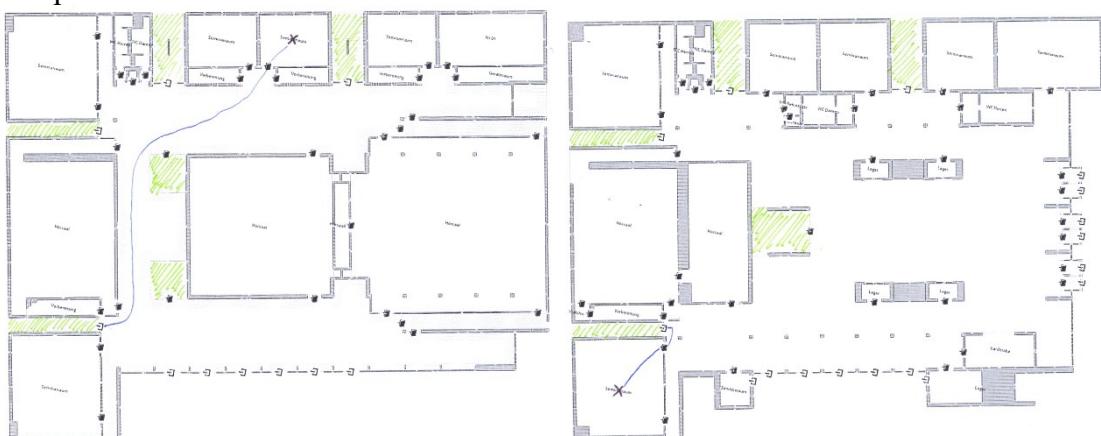
Testperson 7:



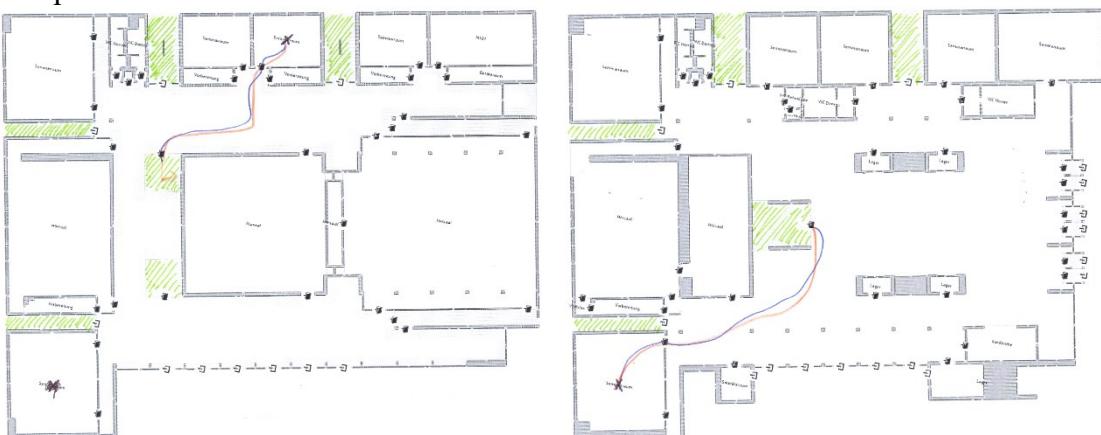
Testperson 8:



Testperson 9:



Testperson 10:

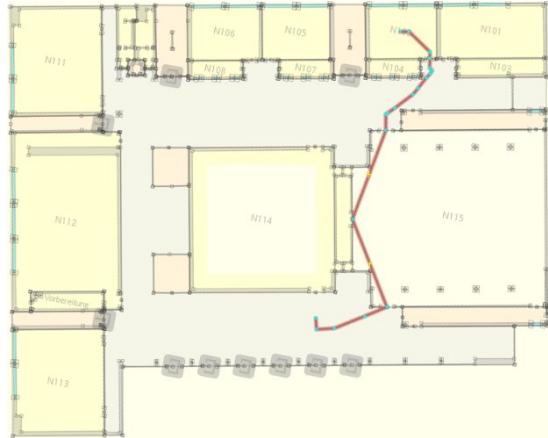


## Durch *GraphHopper* berechnete Routen für die Aufgaben

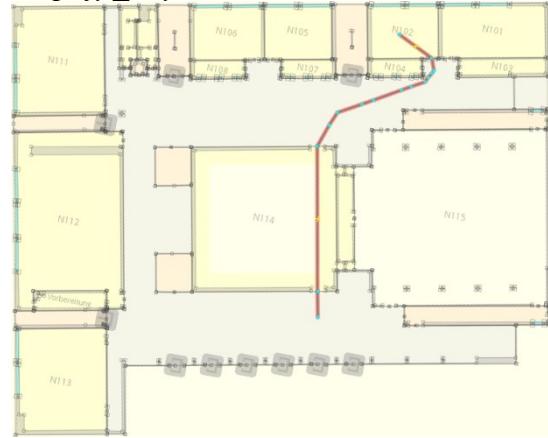
Alle zugrundeliegenden Wegenetzwerke wurden mit Hilfe des *Straight Skeletons* und der Funktion *add\_supplementary* generiert. Sie unterscheiden sich anhand der zusätzlichen Funktionen, die bei der Generierung aufgerufen wurden.

### Aufgabe 1

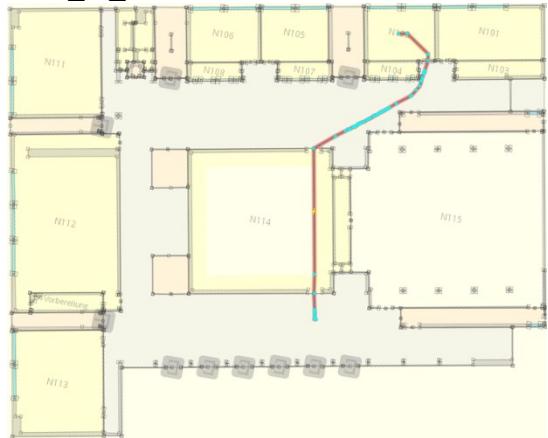
keine zusätzliche Funktionen:



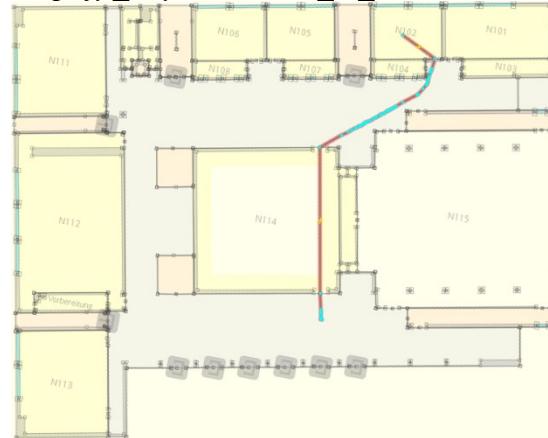
*simplify\_ways*:



*door\_to\_door*:

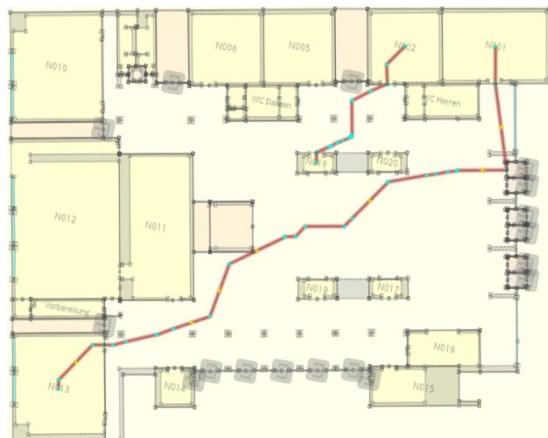


*simplify\_ways* und *door\_to\_door*:

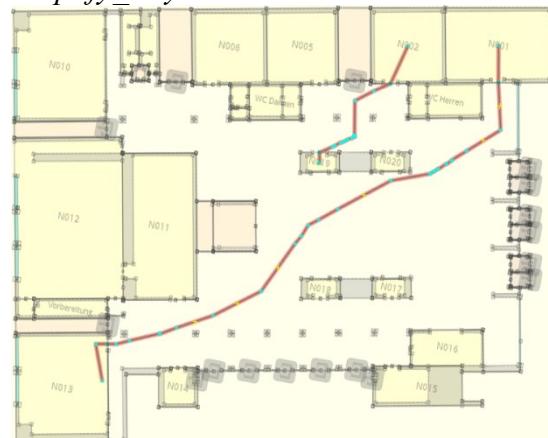


### Aufgaben 2 und 3

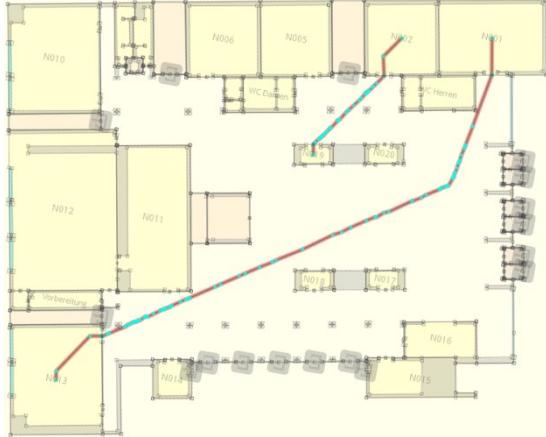
keine zusätzlichen Funktionen:



*simplify\_ways*:



*door\_to\_door:*

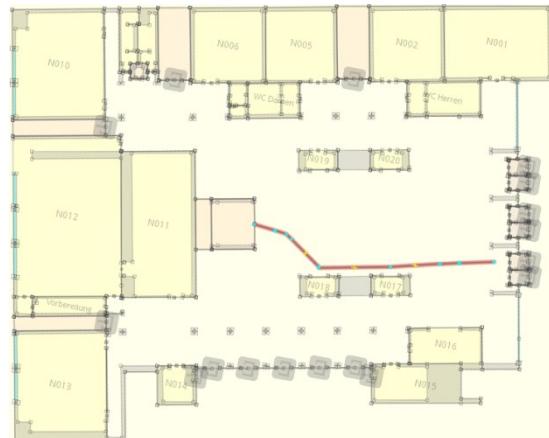
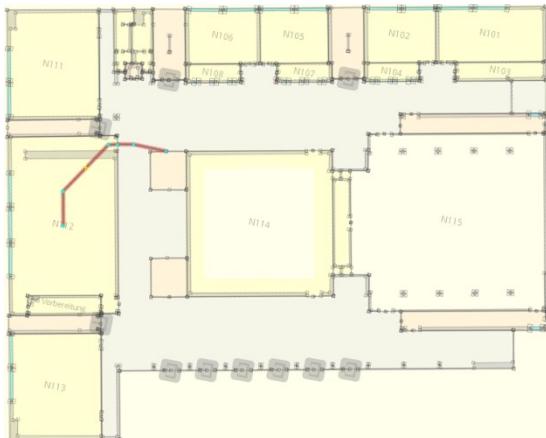


*simplify\_ways und door\_to\_door:*

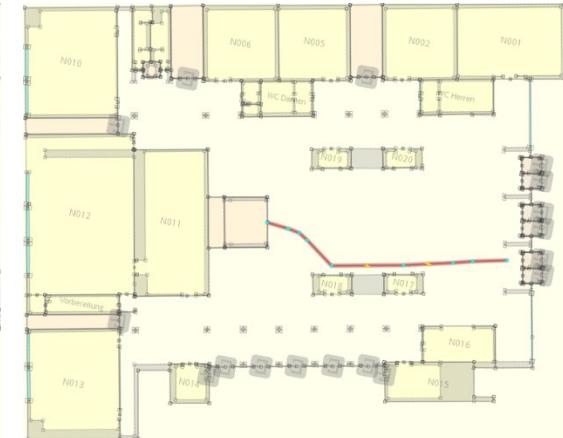
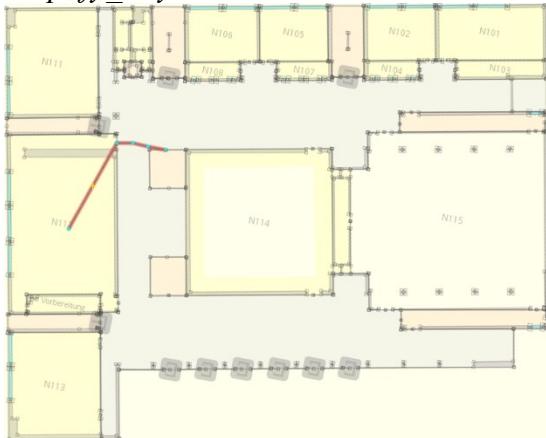


#### Aufgabe 4

ohne zusätzliche Funktionen:



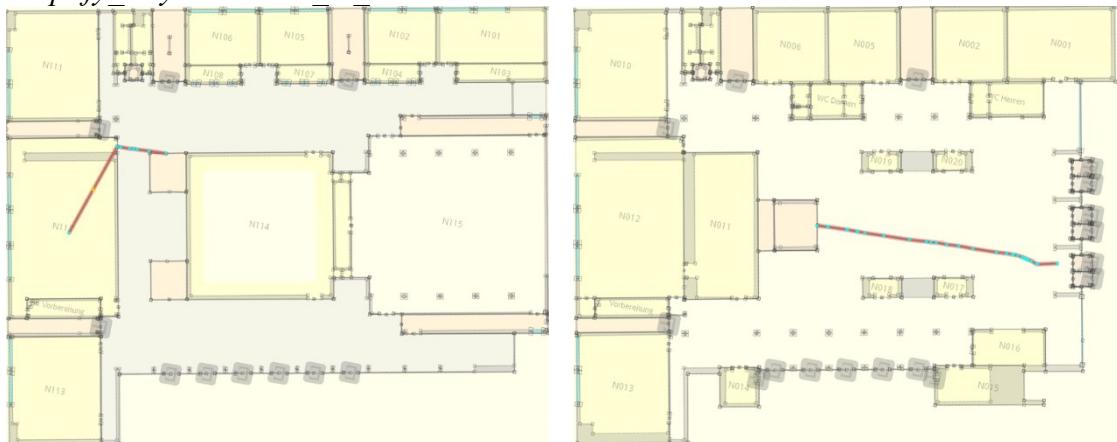
*simplify\_ways:*



*door\_to\_door:*

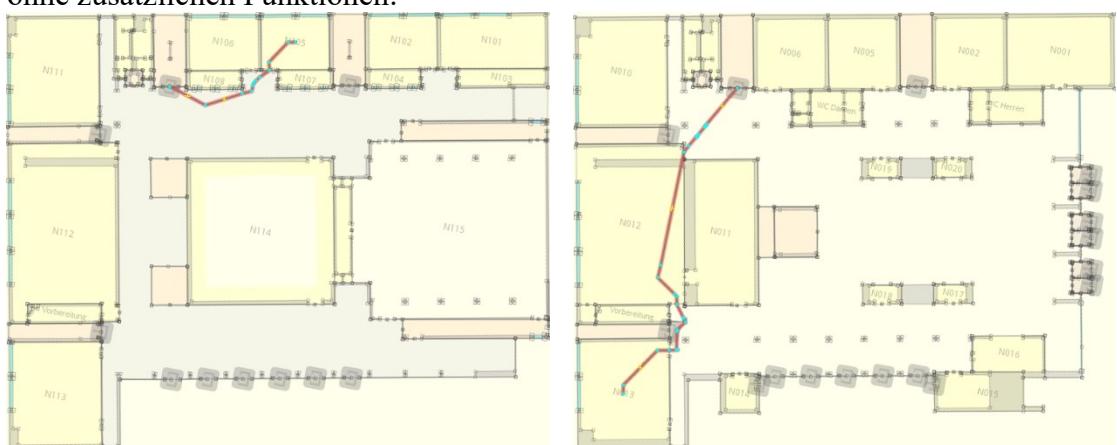


*simplify\_ways und door\_to\_door:*

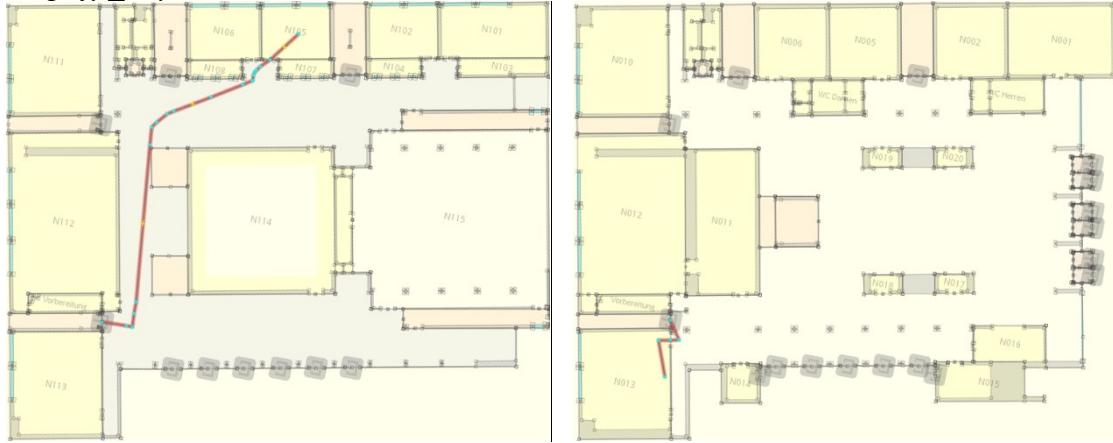


## Aufgabe 5

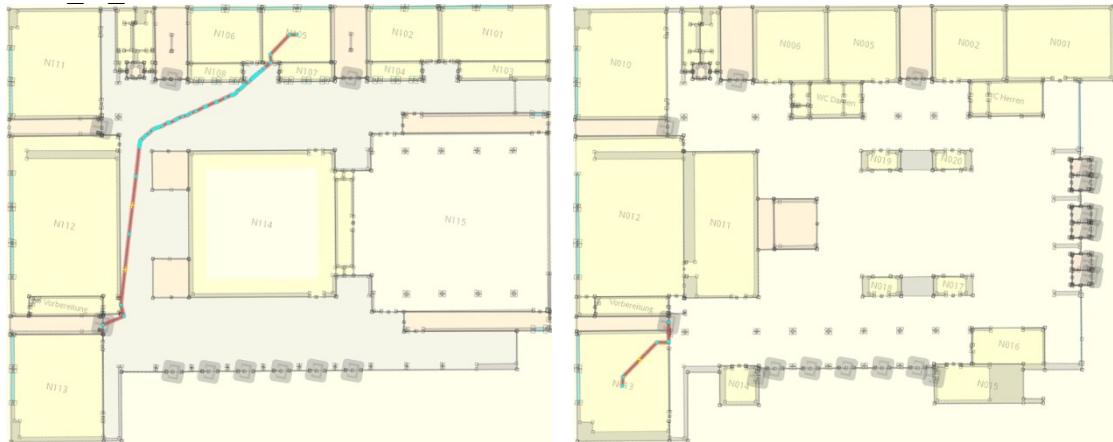
ohne zusätzlichen Funktionen:



*simplify\_ways:*



*door\_to\_door:*



*simplify\_ways* und *door\_to\_door*:

