Praktikum Algorithmen

Wintersemester 2026/27

TECHNISCHE UNIVERSITÄT DARMSTADT

Das Praktikum behandelt Themen im Rahmen des MOTIS-Projekts[1] und erfordert eine selbstständige Einarbeitung in dieses. Unsere Erfahrungen haben gezeigt, dass Studierende die Komplexität eines modernen, real-world C++-Projekts und den damit verbundenen Arbeitsaufwand häufig unterschätzen.

Führen Sie daher bitte **vor** einer Teilnahme die folgenden Schritte zur Selbsteinschätzung Ihrer Eignung durch.

1. Forken Sie das MOTIS-Projekt auf GitHub und klonen Sie Ihren Fork
2. Bauen Sie das Projekt:
   - Linux Developer Setup[2]
   - Windows Developer Setup[3]
   - macOS Developer Setup[4]
3. Setzen Sie entsprechend der Anleitung[5] einen MOTIS-Server auf
4. Rufen Sie `localhost:8080` mit Ihrem Browser auf und prüfen Sie die Funktionalität mit einigen Testanfragen an das Routingsystem

Uns ist bewusst, dass das erstmalige Aufsetzen des Projekts je nach Vorkenntnissen mit einer steilen Lernkurve verbunden sein kann. Die Selbsteinschätzung dient nicht dazu, Studierende von der Teilnahme am Praktikum auszuschließen, sondern soll eine Hilfestellung sein und Sie beim Sicherstellen Ihres Studienerfolges unterstützen. Das selbstständige Einarbeiten in einen unbekannten, umfangreichen Quellcode anhand bereitgestellter Dokumentation ist eine Schlüsselfähigkeit, die Sie im Rahmen dieser Selbsteinschätzung verbessern können.

---

[1] `https://github.com/motis-project/motis`
[2] `https://github.com/motis-project/motis/blob/master/docs/linux-dev-setup.md`
[3] `https://github.com/motis-project/motis/blob/master/docs/windows-dev-setup.md`
[4] `https://github.com/motis-project/motis/blob/master/docs/macos-dev-setup.md`
[5] `https://github.com/motis-project/motis/blob/master/docs/dev-setup-server.md`

Modul: 20-00-0189 Praktikum Algorithmen, 6 CP

Fußgängerrouting mit OSR: Area Routing

OSR (Open Street Router) ist ein speichereffizienter Straßenrouter, der verschiedene Nutzerprofile (z. B. Fußgänger, Fahrrad, Auto) unterstützt. Als Datengrundlage dient OpenStreetMap (OSM).

Das OSR-Repository auf GitHub: `https://github.com/motis-project/osr`

Neben Wegen, die quasi 1:1 als routbarer Graph aufgefasst werden können, enthält OSM auch Flächen wie zum Beispiel Plätze oder, im Kontext des Public Transport Routing für MOTIS besonders wichtig, Bahnhofshallen, Bahnsteige und Korridore in Umsteigebauwerken. Manche von ihnen sind explizit mit wichtigen Wegebeziehungen versehen, andere stellen einfach nur eine Fläche dar, über die ein Mensch nach Belieben laufen kann. OSR ignoriert diese Flächen bisher bzw. läuft nur auf deren Rändern entlang.

Abbildung 1: Zürich HB ohne Area Routing

Für das Area Routing gibt es zahlreiche Ansätze. Die meisten davon basieren darauf, dass in einem Preprocessing-Schritt im Bereich der Flächen zusätzliche Wegebeziehungen generiert und in den Routing-Graph aufgenommen werden. Einige der Ansätze sind z. B. in [Hah+18] und [Gra16] erläutert. Im Rahmen des Praktikums sollen zwei der Ansätze implementiert und verglichen werden.

Zunächst soll der folgende Ansatz als Baseline implementiert werden:

Visibility-Graph [And+15; Gra16]: Die Eckpunkte der Fläche (Außenrand, Löcher und Barrieren, siehe Abschnitt „Flächen“) und die Zugänge (siehe Abschnitt „Zugänge einer Fläche“) dienen als Knoten. Zwei Knoten werden verbunden, wenn sie sich sehen, d. h. wenn die Verbindungsstrecke vollständig innerhalb der Fläche liegt und weder ein Loch noch eine Barriere schneidet. Da kürzeste Wege in einer polygonalen Fläche nur an deren Eckpunkten abknicken, enthält der Visibility-Graph die kürzesten Wege zwischen allen Zugängen, vgl. [Ber+08, Kap. 15]. Genauer knicken kürzeste Wege nur an Ecken ab, an denen der Innenwinkel der

begehbaren Fläche größer als $180°$ ist, also an den nach innen einspringenden Ecken des Außenrands, den nach außen zeigenden Ecken der Löcher und den Eckpunkten der Barrieren. Als Zwischenknoten genügen daher diese Ecken; alle anderen Eckpunkte außer den Zugängen können entfallen. Der Visibility-Graph dient als Baseline für den Vergleich.

Zum Vergleich soll *einer* der folgenden Ansätze implementiert werden:

Grid vs. SpiderWeb [Gra16; Dza+15]: Die freie Fläche wird mit einem Knotengitter gefüllt. Es sollen zwei Varianten implementiert und verglichen werden: ein reguläres Grid, in dem jeder Knoten nur mit seinen direkten Nachbarn verbunden ist [Gra16], und der SpiderWeb-Graph, in dem jeder Knoten mit allen Knoten (inkl. der Zugänge) innerhalb eines festen Radius verbunden wird, sofern die Verbindung weder ein Loch noch eine Barriere schneidet [Dza+15]. Beim regulären Grid müssen die Zugänge in einem eigenen Schritt angebunden werden, z. B. mit den nächstgelegenen Gitterknoten, die sie sehen [Gra16]. Da beide Varianten auf großen Flächen sehr viele Knoten und Kanten erzeugen, soll der Graph anschließend auf kürzeste Wege reduziert werden: Es werden nur die Knoten und Kanten behalten, die auf einem kürzesten Weg zwischen zwei Zugängen der Fläche liegen. Untersucht werden soll, wie Maschenweite bzw. Radius sowie die Reduktion Weglänge, Graphgröße und RAM-Verbrauch beeinflussen.

Triangulierung + Visibility [XWZ16]: Die Fläche wird mit einer (constrained) Delaunay-Triangulierung zerlegt. Aus der Triangulierung werden Knoten abgeleitet. Diese Knoten und die Zugänge werden paarweise verbunden, sofern sie sich sehen. Es sollen zwei Varianten der Knotenwahl implementiert und verglichen werden: die Schwerpunkte der Dreiecke [XWZ16] und die Mittelpunkte der inneren Dreieckskanten, also der Übergänge zwischen benachbarten Dreiecken, über die auch Navigation Meshes ihre Regionen verbinden [Tol+20].

Triangulierung + Triangulierung Die Fläche inkl. punktförmiger Hindernisse (z.B. Brunnen, Bäume) wird mit einer (constrained) Delaunay-Triangulierung zerlegt (vgl. [Hah+18], [XWZ16]). Die Umkreismittelpunkte dieser Dreiecke werden als Knoten für den nächsten Schritt verwendet (entspricht den Voronoi-Knoten). Diese Knoten zusammen mit den Zugängen sind Grundlage für eine weitere (constrained) Delaunay-Triangulierung, deren Kanten dann den Routing-Graph ergeben. Auf diese Weise wird der Abstand von Hindernissen maximiert (im Gegensatz zu einem reinen Delaunay-Ansatz, vgl. [Hah+18]) und gleichzeitig werden die Zugänge direkt angebunden (im Gegensatz zu einem reinen Voronoi-Ansatz, vgl. [Hah+18]).

Medial Axis [Gra16; TCG11; Tol+18]: Die Wege verlaufen mittig zwischen Außenrand, Löchern und Barrieren. Die Medial Axis kann exakt als Voronoi-Diagramm der Randsegmente berechnet werden, vgl. [Ber+08, Abschn. 7.3] und [Syd], oder genähert über das Voronoi-Diagramm der Punkte des verdichteten Randes

bzw. die Delaunay-Triangulierung. Die Zugänge der Fläche müssen in einem eigenen Schritt an die Achse angeschlossen werden; beim reinen Voronoi-Ansatz in [Hah+18] fehlt dieser Schritt.

Andere: Ein anderer Ansatz nach Absprache mit dem Algo-Team.

Die Implementierung erfolgt in C++ direkt in OSR. Die zusätzlichen Knoten und Kanten werden im Extract-Schritt von OSR (`osr::extract`) erzeugt und fest in den Routing-Graphen eingebaut. So können die bestehenden Routing-Algorithmen sie ohne Änderung verwenden. Eingabe des Extract-Schritts ist die unveränderte OSM-Datei. Ein vorgeschaltetes Werkzeug, das die OSM-Datei umschreibt oder zusätzliche Wege einfügt, ist nicht zulässig. Der Ansatz und seine Parameter sollen beim Extract per Option wählbar sein. Ohne Option entsteht der Originalzustand.

Für die Algorithmen dürfen bestehende Bibliotheken genutzt werden (z. B. Boost, tg Geometry). Sie müssen über das Build-System von OSR eingebunden werden und mit dessen MIT-Lizenz vereinbar sein. Start und Ziel einer Anfrage können auch mitten auf einer Fläche liegen. Sie werden durch das bestehende Matching von OSR auf die nächstgelegenen Wege abgebildet (Lot auf das nächste Segment jedes Weges innerhalb eines Suchradius). Da das Matching nur Wege kennt, müssen die neuen Kanten als Wege im Routing-Graphen von OSR angelegt werden. Das Matching selbst wird als Black Box betrachtet und ist nicht Teil der Aufgabe; der Fokus liegt auf dem Routing zwischen den Zugängen einer Fläche. Neben den Flächen selbst müssen auch Gebäude und Barrieren berücksichtigt werden, die auf ihnen stehen. Welche OSM-Objekte als Flächen berücksichtigt werden, legt Abschnitt „Flächen“ fest.

Die beiden implementierten Ansätze (Visibility-Graph und der gewählte zweite Ansatz) sollen verglichen werden. Verlangt der gewählte Ansatz zwei Varianten, werden beide Varianten jeweils mit dem Visibility-Graphen und untereinander verglichen. Der Vergleich erfolgt hinsichtlich der Preprocessing-Laufzeit, der Anzahl erstellter Knoten und Kanten, des zusätzlichen RAM-Verbrauchs für den Routing-Graphen, des Einflusses auf die Routing-Zeiten von zufällig generierten Queries sowie der Abweichung der gefundenen kürzesten Wege zwischen dem Originalzustand, dem Visibility-Graphen (Optimum) und dem gewählten Ansatz. Die Experimente sollen mindestens auf dem OSM-Extract der Schweiz durchgeführt werden (`https://download.geofabrik.de/europe/switzerland.html`).

Die subjektive Qualität der berechneten Wege kann ebenso beleuchtet werden, allerdings ist es durchaus denkbar, dass für die Anzeige von „schönen“ Wegen ein völlig anderer, komplexerer Algorithmus verwendet wird, der nicht Teil des Praktikums ist. Dann stellt das Routing nur eine möglichst gute Annäherung an den kürzesten, aber auch „schönsten“ Weg dar.

Flächen

Eine *Fläche* ist ein geschlossener Weg oder eine Multipolygon-Relation aus OSM, die Fußgänger in beliebiger Richtung überqueren können. Berücksichtigt werden Objekte mit einem der folgenden Tags:

- `highway` gleich `pedestrian`, `footway`, `path`, `service`, `living_street`, `corridor`, `platform` oder `steps`,
- `public_transport=platform` oder `railway=platform`.

Ein geschlossener Weg ist nur mit `area=yes` eine Fläche. Ohne dieses Tag ist er ein ringförmiger Weg, z. B. ein Fußweg um einen Teich. Eine Multipolygon-Relation ist immer eine Fläche.

Keine Flächen sind:

- Objekte mit `area=no`.
- Objekte mit `public_transport=station`. Sie umfassen einen ganzen Bahnhof über alle Ebenen, eine Überquerung würde zwischen den Ebenen springen.
- Objekte, die nur `place=square` tragen. Das Tag benennt einen Platz, sagt aber nicht, wo man gehen kann. Über manche solcher Plätze führt eine Straße.

Hat ein Multipolygon mehrere äußere Ringe, bildet jeder davon mit seinen inneren Ringen eine eigene Fläche.

Die *Löcher* einer Fläche sind ihre inneren Ringe (`inner`) und die Gebäude, die auf ihr stehen. Ein *Gebäude* ist ein geschlossener Weg mit `building`, außer `building=no` und `building=roof`. Letzteres ist ein Dach auf Stützen, z. B. über einer Tankstelle oder einem Bahnsteig, unter dem man gehen kann. Ein Gebäude steht auf einer Fläche, wenn mindestens einer seiner Knoten im Inneren der Fläche liegt (nicht auf dem Rand) und seine `level` die Ebene der Fläche enthalten. Sein `layer` spielt keine Rolle, denn es ordnet das Gebäude nur gegenüber Tunneln und Brücken ein. Ein Gebäude mit `building:min_level` > 0 oder `min_height` $> 2{,}5\,\mathrm{m}$ steht auf Stützen und blockiert nur Flächen, die auf seinen Ebenen liegen, nicht den Boden darunter.

Innere Ringe und Gebäude überlappen sich oft, z. B. wenn ein Gebäude zugleich als innerer Ring erfasst ist oder über den Außenrand hinausragt. Die Löcher werden daher vereinigt und auf den Außenrand zugeschnitten. Ein Loch in einem Loch, etwa ein Innenhof in einem Gebäude, ist von der Fläche aus nicht erreichbar und entfällt. Ungültige Polygone werden ignoriert.

Barrieren sind Wege, die auf derselben Ebene (`level`) und demselben `layer` wie die Fläche liegen und `barrier` gleich `yes`, `wall`, `fence`, `hedge`, `retaining_wall`, `city_wall` oder `guard_rail` tragen. Ein Zaun auf einer Brücke blockiert den Platz darunter nicht. Barrieren sind Linien, keine Löcher: Keine neue Kante darf eine Barriere schneiden. Es zählt jedes Segment einer Barriere, von dem mindestens ein Endpunkt

im Inneren der Fläche liegt, auch wenn es den Rand kreuzt. Eine Barriere, die nur auf dem Rand verläuft, ist Teil des Randes und kein Hindernis.

Dieselbe Fläche ist in OSM oft mehrfach erfasst, z. B. ein `place=square` über der `highway=pedestrian`-Fläche oder eine kleine Fläche um einen Bahnhofseingang innerhalb eines Platzes. Flächen mit denselben `level`- und `layer`-Werten, deren Inneres sich überlappt, werden daher zu einer Fläche vereinigt. Flächen, die sich nur am Rand berühren, bleiben getrennt. Sie sind über gemeinsame Zugänge verbunden. Die Anzahl der vereinigten Flächen soll in der Auswertung angegeben werden.

Zugänge einer Fläche

Zugänge einer Fläche sind die Stellen, an denen das bestehende Wegenetz die Fläche erreicht. Dafür zählen nur Wege, die für Fußgänger routbar sind und auf derselben Ebene wie die Fläche liegen. Ausgenommen davon sind Treppen, Rampen und Aufzüge (siehe unten). Der Rand der Fläche selbst zählt nicht als solcher Weg. Abbildung 2 zeigt ein Beispiel.

Ein Knoten eines solchen Weges ist ein Zugang, wenn er

- auf dem Rand der Fläche liegt, also auch zum Außenrand oder zu einem Loch gehört (●), oder
- im Inneren der Fläche liegt (■). Das betrifft z. B. Wege, die auf der Fläche enden oder sie überqueren.

Treppen (`highway=steps`), Rampen (Wege mit `ramp=yes` oder `incline`) und Aufzüge (`highway=elevator`, als Knoten oder Weg) verbinden Ebenen. Sie tragen daher oft mehrere `level`-Werte (z. B. `level=-1;0`) oder gar keinen. Für sie gilt das Ebenen-Kriterium nicht. Stattdessen wird für jeden ihrer Knoten geprüft, ob das Polygon der Fläche ihn enthält (Rand eingeschlossen, Löcher ausgenommen). Jeder so enthaltene Knoten ist ein Zugang (● bzw. ■), unabhängig von `level`, `layer` oder `tunnel`.

Keine Zugänge sind:

- Eckpunkte des Randes, die zu keinem anderen Weg gehören (○).
- Knoten von Wegen auf einer anderen Ebene (×). Das sind Tunnel, Brücken und Wege, deren `level` die Ebene der Fläche nicht enthält, sofern es sich nicht um Treppen, Rampen oder Aufzüge handelt.
- Stellen, an denen ein Weg den Rand kreuzt, ohne einen Knoten mit ihm zu teilen. Solche Kreuzungen müssen nicht berücksichtigt werden. Ihre Anzahl soll aber in der Auswertung angegeben werden.

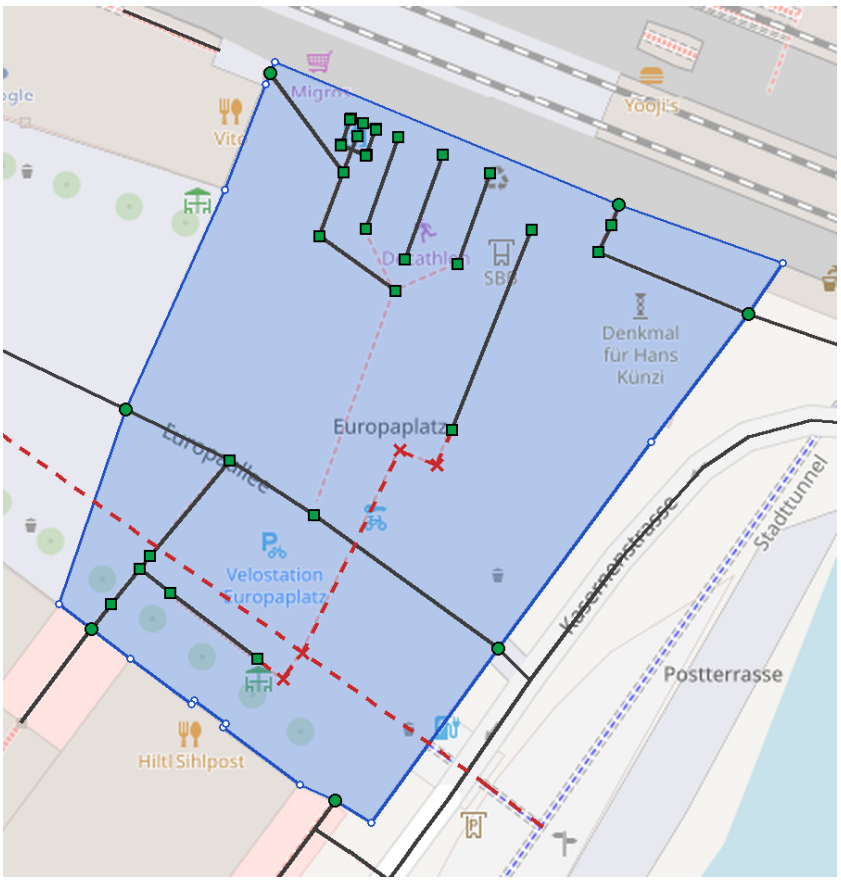

Abbildung 2: Zugänge des Europaplatzes in Zürich (blau). Zugänge auf dem Rand: ●; Zugänge im Inneren: ■; Randknoten ohne Zugang: ○; Tunnel und Wege auf Ebene -1 (gestrichelt) liefern keine Zugänge: ×. Treppen, Aufzüge und die Rampe in den Tunnel liefern dagegen über den Polygon-Test Zugänge, auch wenn sie auf Ebene -1 liegen. Karte: © OpenStreetMap-Mitwirkende (ODbL).

Die neuen Kanten müssen an genau diese bestehenden Knoten angeschlossen werden. An derselben Position dürfen keine doppelten Knoten entstehen, sonst ist der Routing-Graph über die Fläche hinweg nicht zusammenhängend. Hat eine Fläche weniger als zwei Zugänge, werden für sie keine Kanten erzeugt. Auch die Anzahl solcher Flächen soll in der Auswertung angegeben werden.

Organisatorisches

Alles Organisatorische findet im Moodle-Kurs statt.

Abgaben

- Alle Abgaben finden über Moodle statt.
- Alle Abgaben inkl. der Fortschrittsberichte sind **zum Bestehen erforderlich**.
- Alle Deadlines in Moodle sind strikt. Es gibt keine versteckte spätere Deadline, zu der Abgaben noch akzeptiert werden.

Weitere Hinweise

- Eine Bearbeitung des Praktikums in Gruppen ist nicht möglich.
- Obige Aufgabenstellung betrifft das Modul 20-00-0189 Praktikum Algorithmen. Nach erfolgreichem Abschluss ist es möglich, in einem späteren Semester das Modul 20-00-0276 Praktikum Algorithmen II zu belegen, in dem dann das jeweils aktuelle Thema des Praktikums Algorithmen I bearbeitet werden kann, d. h. dies dient i. d. R. nur dazu, das Praktikum erneut belegen zu können.
- Das Modul 20-00-1029 Projektpraktikum Algorithmik kann nur nach erfolgreichem Abschluss des Moduls 20-00-0189 Praktikum Algorithmen oder alternativ einer Bachelor-/Masterarbeit am Fachgebiet Algorithmik belegt werden. Hierbei werden mit den Studierenden individuelle Aufgabenstellungen erarbeitet.
- Planen Sie bitte zusätzlichen Aufwand ein, falls die verwendete Programmiersprache C++ oder die verwendeten Technologien (z. B. Git, CMake) neu für Sie sind.
- Als Referenz für die Komplexität des Projektes, das im Praktikum erweitert wird, dient MOTIS.
  Bitte überlegen Sie sich genau, ob Sie mit einer großen real-world Modern-C++-Codebasis klarkommen. Bisherige Erfahrungen haben gezeigt, dass Studierende den Einarbeitungsaufwand und die Komplexität der Programmiersprache unterschätzen.
- Einen Crashkurs in die MOTIS-typische C++-Programmierung gibt es hier: `https://www.algo.informatik.tu-darmstadt.de/dl/cpp.pdf`.
- MOTIS C++ Style Guide: `https://github.com/motis-project/motis/blob/master/docs/STYLE.md`

Literatur

[And+15] Simeon Andreev u. a. „Towards Realistic Pedestrian Route Planning“. In: *15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2015)*. Hrsg. von Giuseppe F. Italiano und Marie Schmidt. OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015, S. 1–15. DOI: 10.4230/OASICS.ATMOS.2015.1.

[Ber+08] Mark de Berg u. a. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, 2008. ISBN: 9783540779742. DOI: 10.1007/978-3-540-77974-2.

[Dza+15] Dzenan Dzafic u. a. „Routing über Flächen mit SpiderWebGraph“. In: *AGIT – Journal für Angewandte Geoinformatik* 1 (2015), S. 516–525. DOI: 10.14627/537557071.

[Gra16] Anita Graser. „Integrating Open Spaces into OpenStreetMap Routing Graphs for Realistic Crossing Behaviour in Pedestrian Navigation“. In: *GI_Forum* 4.1 (2016), S. 217–230. ISSN: 2308-1708. DOI: 10.1553/giscience2016_01_s217.

[Hah+18] Stefan Hahmann u. a. „Routing through open spaces – A performance comparison of algorithms“. In: *Geo-spatial Information Science* 21.3 (2018), S. 247–256. DOI: 10.1080/10095020.2017.1399675.

[Syd] Andrii Sydorchuk. *Boost.Polygon Voronoi Extensions*. Boost C++ Libraries. URL: https://www.boost.org/doc/libs/latest/libs/polygon/doc/voronoi_main.htm (besucht am 23. 09. 2026).

[TCG11] W. van Toll, A. F. Cook und R. Geraerts. „Navigation meshes for realistic multi-layered environments“. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sep. 2011, S. 3526–3532. DOI: 10.1109/iros.2011.6094790.

[Tol+18] Wouter van Toll u. a. „The medial axis of a multi-layered environment and its application as a navigation mesh“. In: *ACM Transactions on Spatial Algorithms and Systems (TSAS)* 4.1 (2018), S. 1–34. DOI: 10.1145/3204456.

[Tol+20] Wouter van Toll u. a. „Comparing navigation meshes: Theoretical analysis and practical metrics“. In: *Computers & Graphics* 91 (2020), S. 52–82. ISSN: 0097-8493. DOI: 10.1016/j.cag.2020.06.006.

[XWZ16] Man Xu, Shuangfeng Wei und Sisi Zlatanova. „An Indoor Navigation Approach Considering Obstacles and Space Subdivision of 2D Plan“. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLI-B4 (Juni 2016), S. 339–346. ISSN: 2194-9034. DOI: 10.5194/isprs-archives-xli-b4-339-2016.