Fakultät Bauingenieurwesen Institut für Bauinformatik

DIPLOMARBEIT

Navigation von mobilen Robotern für den Innenausbau von Gebäuden

Navigation of Mobile Robots for Interiorer Constructions Tasks

eingereicht von

cand. ing. Eric Kaulfuß
geb. am 12.01.1994 in Dresden
Matrikel-Nummer: 3855042

Betreuer:

- Dipl.-Ing. Nicolas Mitsch

Mentoren:

- Dr.-Ing. Katja Heine (BTU Cottbus-Senftenberg)
- Dipl.-Ing. Martin Jahn (Ed. Züblin AG)

Verantwortlicher Hochschullehrer:

- Prof. Dr.-Ing. habil. Karsten Menzel

Zweitprüfer:

- Prof. Dr.-Ing. Raimar Scherer

Dresden, den 21.09.2021

TECHNISCHE
UNIVERSITÄT
DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

Aufgabenstellung für die Diplomarbeit

Name: cand. ing. Eric Kaulfuß (Matrikel Nr.: 3855042)
Vertiefung: Computational Engineering

Thema: **Navigation von mobilen Robotern für den Innenausbau von Gebäuden**
Navigation of Mobile Robots for Interior Construction Tasks

Zielsetzung der Diplomarbeit:

Um Roboter auf Baustellen effektiv einsetzen zu können, müssen diese möglichst flexibel und mit einem hohen Grad an Autonomie eingesetzt werden können. Andererseits sollten Roboter für das Bauwesen kostengünstig, d.h. mit einem begrenzten Umfang an IT-Komponenten entworfen sein. Damit sind zwei wesentliche Randbedingungen für den Entwurf von Navigationssystemen für Bauroboter vorgegeben.

Verfügbare Informationen zur initialen Ermittlung von Fahrwegen für Bauroboter können aus Gebäudeplänen ermittelt werden. Dabei ist jedoch zu beachten, dass eine Aktualisierung der Gebäudepläne in der Bauphase in Echtzeit nicht möglich ist. Weiterhin ist zu beachten, dass Navigationssysteme wie GPS für Ausbaugewerke nicht oder nur bedingt nutzbar sind. Deshalb müssen Robotersteuerungen über ein gewisses Maß an Intelligenz verfügen, um sich auf Baustellen zu orientieren. Diese zusätzlichen informationen müssen aus anderen Navigationshilfen, wie z.B. Mustererkennung von eingemessenen Marken oder Sensoren ermittelt werden.

Als erste Grundlage zur Steuerung von Robotern auf Baustellen könnten zunächst aus BIM-Modellen erzeugte Graphen angesehen werden. Diese Graphen sollen ein Grundgerüst an Fahrwegen bereitstellen, das zur Navigation des Roboters zwischen Fixpunkten (z.B. Lagerplatz, Einsatzort, Ladestation, oder temporär nicht zugängliche Orte) herangezogen wird.

Zahlreiche, ergänzende Positionierungstechnologien stehen zur Verfügung. Diese sind auf ihre Kombinationsfähigkeit mit BIM-Technologien zu untersuchen. Schlussendlich ist im Rahmen der Diplomarbeit ein Konzept für ein baustellentaugliches, preiswert umzusetzendes, möglichst BIM-basiertes Navigationssystem zu entwickeln und prototypisch zu implementieren.

Diplomarbeit:

Im Rahmen der Ausarbeitung sollen die folgenden Punkte bearbeitet werden:

1. Recherche zur Graphenerzeugung aus BIM-Modellen und Aufbereitung der Ergebnisse.
2. Recherche zu Schnittstellen zur Übergabe von BIM-basierten Navigationsgraphen an mobile Roboter.
3. Recherche zu weiterführenden, baustellentauglichen Navigationshilfen für Baurobotor.
4. Diskussion und Bewertung von Lösungsmöglichkeiten zur genauen Positionierung von autonomen Baurobotern in Gebäuden mit einer Toleranz von ca. 5 .. 10 cm.
5. Verifizierung der Ergebnisse durch Implementierung eines selbstgewählten Beispiels.

Verantwortliche und Termine:

Verantwortlicher Hochschullehrer und Erstprüfer	Prof. Dr.-Ing. habil. Menzel
Zweitprüfer	Prof. Dr.-Ing. Raimar Scherer
wiss. Betreuer	Dipl.-Ing. Nicolas Mitsch
Mentoren	Dr.-Ing. Katja Heine (BTU Cottbus-Senftenberg)
	Dipl.-Ing. Martin Jahn (Ed. Züblin AG)

ausgehändigt am 27.04.2021

einzureichen am 27.08.2021

Prof. Dr.-Ing. habil. K. Menzel
Verantwortlicher Hochschullehrer

SELBSTSTÄNDIGKEITSERKLÄRUNG

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich reiche sie erstmals als Prüfungsleistung ein. Mir ist bekannt, dass ein Betrugsversuch mit der Note „nicht ausreichend“ (5,0) geahndet wird und im Wiederholungsfall zum Ausschluss von der Erbringung weiterer Prüfungsleistungen führen kann.

Name: Kaulfuß

Vorname: Eric

Matrikelnummer: 3855042

Dresden, den 21.09.2021

Unterschrift cand. ing. E. Kaulfuß

I ABSTRACT

Ziel dieser Arbeit ist es, mögliche Varianten zur Lokalisierung und Navigation eines autonomen, mobilen Roboters in einem neu zu errichtenden Gebäude während des Innenausbaus zu untersuchen und eine Variante als Simulation zu Implementieren. Wesentliche Randbedingung ist das Vorhandensein eines vollständigen, dreidimensionalen BIM-Modells des Gebäudes, aus dem Navigationshilfen für den Roboter in Form einer Karte erzeugt werden können.

Dazu wurden zunächst Besonderheiten und Herausforderungen der Baubranche bezüglich des Einsatzes von Robotern auf Baustellen analysiert, mögliche Navigationsmethoden recherchiert und bewertet sowie die Datenübertragung zum Roboter untersucht. Weiterhin fand eine Untersuchung verschiedener Ansätze zur Kartenerzeugung aus BIM-Modellen statt. Der grafische beziehungsweise geometrische Ansatz zur Kartenerzeugung wurde mithilfe mehrerer Simulation in der Umgebung ROS (Robot Operating System) validiert.

II INHALTSVERZEICHNIS

III ABBILDUNGSVERZEICHNIS

IV TABELLENVERZEICHNIS

V ABKÜRZUNGSVERZEICHNIS

AMCL *Adaptive Monte Carlo Localization*
BCF *BIM Collaboration Format*
BIM *Building Information Modeling*
BSD *Berkeley Software Distribution*
CCD *Charge-coupled device (lichtempfindliche Bauteile eines Kamerachips))*
CNN *Cellular Neural Network*
COLLADA Collaborative Design Activity
FMCW *frequency modulation continuous wave*
GNSS *Global Navigation Satellite System*
IFC *Industry Foundation Classes*
JSON *JavaScript Object Notation*
LADAR *Laser Detection and Ranging*
LIDAR *Light detection and ranging*
LTS *Long Term Support*
MAF *Medial Axis Function*
MAT *Medial Axis Transformation*
MEMS *Micro-Electro-Mechanical-Systems*
M-MAT *Modified Medial Axis Transformation*
OCCT *Open CASCADE Technology*
OSRF *Open Source Robotics Foundation*
PDF *Portable Document Format*
PGM *Portable Grey Map*
RFID *Radio Frequency Identification*
ROS *Robot Operating System*
SDF *Simulation Description Format*
SLAM *Simultaneous Localization and Mapping*
S-MAT *Straight Medial Axis Transformation*
SPF *STEP Physical File*
STAIR *Stanford AI Robot*
STEP *Standard for the exchange of product model data*
UWB *Ultra-Wideband*
VDI *Verein Deutscher Ingenieure*
WLAN *Wireless Local Area Network*
XML *Extensible Markup Language*
YAML *YAML Ain’t Markup Language*

1 EINLEITUNG

Durch eine Vielzahl von ökonomischen, sozialen und politischen Veränderungen weltweit kommt es immer häufiger zu personellen Engpässen im Baugewerbe. Insbesondere in Deutschland sind durch demografische Veränderungen immer weniger Arbeitskräfte bei gleichbleibend hohem Bedarf verfügbar. Weiterhin ist auf organisatorischer Ebene im Bereich der Planung und Ausführung ein erhöhtes Fehlerpotential durch mangelnde Kompatibilität der Software, schlecht dokumentierte und händisch ausgeführte, teilweise sehr spontane Umplanung sowie Interpretationsfehler beim Lesen der Baupläne vorhanden. Durch zunehmende Nutzung von Building Information Modeling (BIM) werden die beiden letztgenannten Problemfelder bereits adressiert. Durch gut ausgearbeitete BIM-Modelle lassen sich kollaborative Planung, Datenaustausch und Änderungen sicher und effizient umsetzen. In einem solchen zentralen BIM-Modell sind dadurch auch Informationen gesammelt, die die Anwendung von Robotern im Innenausbau maßgeblich unterstützen können. Diese Diplomarbeit untersucht, wie eine konkrete praktische Umsetzung der Nutzung von BIM-Modellen für den Einsatz von Robotern im Innenausbau aussehen kann.

In diesem Kapitel werden Motivation und Ziel, das Vorgehen und die verwendete Soft- und Hardware der Diplomarbeit vorgestellt.

1.1. MOTIVATION UND ZIEL

In der Bauindustrie wird sowohl in der Planungsphase als auch in der Ausführungsphase nach wie vor sehr traditionell gearbeitet. Verschiedene Akteure im Planungs- und Bauprozess arbeiten relativ abgeschottet voneinander mit eigener, meist proprietärer Software. Der Austausch zwischen den Akteuren erfolgt auch heute (2021) noch überwiegend mit teilweise inkompatiblen Datenstrukturen und -formaten oder sogar nur über PDF-Dokumente beziehungsweise gedruckte 2D-Pläne (Follini et al., 2021). Dadurch kommt es zu Konvertierungs-, Lese- und Änderungsfehlern während des gesamten Prozesses, was Zeit und Ressourcen kostet. Im Vergleich zu anderen Industriezweigen hat die Bauindustrie eine stark ineffiziente Wertschöpfungskette. Das bedeutet, dass der Ertrag, verglichen mit dem dafür nötigen Aufwand, gering ist. Dies liegt unter anderem an mangelnder Digitalisierung bei gleichzeitig immer komplexer werdenden Projekten. Die traditionelle Arbeitsweise und Kommunikation via Papier und zweidimensionaler Planunterlagen ist den gestiegenen Anforderungen nicht mehr gewachsen. Daher wird in den letzten Jahren vermehrt auf die flächendeckende Einführung von BIM-Modellen gedrängt. Diese ermöglichen zentrale Gebäudemodelle und Wissenssammlungen, die von allen Akteuren für Planung, Ausführung und Betrieb von Gebäuden genutzt werden können. Auch der Einsatz von weitestgehend autonom arbeitenden Robotern profitiert von der Einführung von BIM im gesamten Prozess, da Informationen nach entsprechender Umwandlung vom Roboter genutzt werden können. Dadurch sollen zukünftig Roboter relativ autonom und qualitativ ausreichend im Innenausbau arbeiten und damit zunehmend fehlende Arbeitskräfte ersetzen können. Auch das Ausführen von für Menschen potentiell gefährlichen Arbeiten ist denkbar (Agarwal et al., 2016; Scherer & Schapke, 2014).

1.2. AUFBAU DER DIPLOMARBEIT

Im ersten Teil der Diplomarbeit werden theoretische Grundlagen zu verwendeten Konzepten und Begriffen betrachtet. Auch das für den Praxisteil verwendete Robot Operating System (ROS) wird vorgestellt, um die Arbeitsschritte verständlicher zu machen. Anschließend wird näher auf die Möglichkeiten beziehungsweise technischen Lösungen zur Lokalisierung und Navigation eines autonomen Roboters eingegangen. Bestehende Lösungsansätze aus bisherigen Arbeiten werden vorgestellt und auch die Übergabemöglichkeiten der Informationen an den Roboter werden betrachtet.

Der zweite Teil stellt den Praxisteil dar. In diesem wird die Simulation einer Navigationsaufgabe in einer virtuellen Umgebung mithilfe von ROS beschrieben und Ergebnisse erläutert und bewertet. Für die Implementierung des Beispiels wurden digitale Modelle zweier Beispielgebäude genutzt, die ebenfalls im zweiten Teil der Arbeit vorgestellt werden.

1.3. FORMALE HINWEISE

Die in der vorliegenden Arbeit genutzten Fachbegriffe stammen häufig aus dem Englischen. Da sie als feststehende Fachbegriffe auch häufig im Deutschen identisch verwendet werden, werden diese möglichst nicht übersetzt.

Eigennamen sowie Fachbegriffe sind *kursiv* gedruckt. Auf die im Bereich des Ingenieurwesens häufig genutzte Kapitälchenschreibweise von Personennamen wurde dagegen aus Lesbarkeitsgründen verzichtet. Die in der Diplomarbeit gewählte männliche Form (generisches Maskulinum) bezieht sich immer zugleich auf Personen jeglichen Geschlechts. Auf eine Doppelbezeichnung oder ähnliches wurde ebenfalls zugunsten einer besseren Lesbarkeit verzichtet.

2 THEORETISCHE EINBETTUNG

In diesem Kapitel werden die Rahmenbedingungen sowie grundlegende theoretische Konzepte und Definitionen erläutert, die im Kontext dieser Diplomarbeit relevant sind. Dazu wird zu Beginn kurz aufgelistet, welche Wissensbereiche dabei abgedeckt werden müssen.

Damit ein Roboter sinnvoll im Innenausbau praktisch einsetzbar ist, sollte er grundsätzlich folgende Fähigkeiten aufweisen (Palacz et al., 2019, S. 655):

- Informationen aus der Umgebung aufnehmen können
- Längere Zeit autonom arbeiten können
- Bewegung und Orientierung ohne menschliche Hilfe ausführen können

Zusätzlich sollte er geometrische und semantische Informationen aus einem BIM-Modell (siehe Abschnitt 2.3.1) verarbeiten sowie mit seinen eigenen Beobachtungen verknüpfen können. Daher soll folgender, schrittweiser Ablauf für die Umsetzung in der Diplomarbeit genutzt werden:

1. Umwandlung des Wissens im BIM-Modell in eine nutzbare Repräsentation (als Graph, Topologie, Bild oder ähnliches)
2. Übergabe der Wissensrepräsentation an Roboter
3. Navigationspfade finden
4. Positionierung und Wegfindung des Roboters aus
    - Wissen aus BIM-Modell
    - Scan der Umgebung (Kameras, Lidar etc.) und Verknüpfung der BIM-Daten mit eigenen Beobachtungen
    - Eventuell zusätzliche Lokalisierung durch WLAN, Bilderkennung o.ä.

2.1. UNTERSCHIEDE ZU ANDEREN INDUSTRIEZWEIGEN

Der Einsatz von Robotern in der Fertigung ist in anderen Industriezweigen wie Automobil- oder Flugzeugindustrie seit Jahren Standard (Siciliano & Khatib, 2016). In der Baubranche herrscht seit den letzten zwei Jahrzehnten ein wachsendes Interesse für Roboter im Bauwesen, um die in der Einleitung bereits genannten Nachteile auszugleichen. Schon in den siebziger Jahren wurden Untersuchungen zum Robotereinsatz im Bereich der Ausführung von Schwerlastarbeiten wie beispielsweise Forst-, Erd- oder Montagearbeiten von großen Außenfassaden durchgeführt. Auch robotergestützte Dokumentation des Baufortschritts sowie Inspektion und Wartung wurden Ziele der Forschung. Erst durch das Aufkommen von BIM-Modellen und gleichzeitig starker Weiterentwicklung der Robotertechnik vor allem im Bereich der Lokalisation und Navigation in Innenräumen konnte in den letzten Jahren auch der Inneneinsatz mobiler Roboter in Betracht gezogen werden. Daher beziehen neue Untersuchungen zunehmend BIM-Modelle als Informationsquelle für Steuerung, Lokalisierung und Wegfindung eines Roboters mit ein (Carra et al., 2018, S. 1–2).

Im Gegensatz zu Baustellen herrschen in Fabrikhallen allerdings klar definierte Umgebungen. Die Geometrie, Topologie und Semantik einer Fabrikhalle ist relativ statisch und kann von Anfang an für den Roboter vorteilhaft geplant werden. Navigationshilfen, notwendiger Platz und Interaktion mit der Umgebung und anderen Menschen oder Maschinen sind meist standardisiert. Für einen geplanten Einsatz im Baugewerbe und vor allem im Innenausbau ergeben sich damit verglichen mit herkömmlichen Industrierobotern vor allem folgende Probleme:

- Lokalisierung des Roboters in Innenräumen mit GPS nicht möglich
- Keine Orientierungshilfen am Boden möglich
- Teilweise stark veränderliche Umgebungen auf Baustellen durch Baufortschritt, temporäre Lagerflächen und damit auch temporäre Unzugänglichkeit von Durchgängen oder Wegen
- Einzigartigkeit jedes Gebäudes in Geometrie und Topologie

Diese Probleme unterscheiden den Einsatz im Baugewerbe stark von herkömmlichen, bereits etablierten Anwendungsbereichen von Robotern und stellen besondere Herausforderungen in Planung und Umsetzung dar (Carra et al., 2018, S. 2).

2.2. VORTEILE DER INTEGRATION VON BIM

Durch den Einsatz von BIM-Modellen für den Einsatz von Robotern im Baugewerbe ergeben sich folgende Vorteile (Hamieh et al., 2017; Kerbitz, 2019; F. Lin et al., 2019; Wei & Akinci, 2018):

- Die Wegfindung durch Gebäude ist in nahezu vollständig vorhandenen Gebäudemodellen schneller und effizienter als in vom Roboter ausschließlich selbst erstellten Karten
- Die zusätzliche Übergabe von semantischen Gebäudeinformationen wie Raumnutzungsart oder temporär vorhandene Hindernisse (zum Beispiel während eines bestimmten Bauabschnittes vorhandene Bodenöffnungen oder zeitweise unzugängliche Räume) ermöglicht eine effizientere und sicherere Wegplanung
- Informationen über Änderungen an Gebäudegeometrie und -topologie können stunden- oder tagesaktuell für den Roboter verfügbar gemacht werden, um diese in der Wegplanung berücksichtigen zu können

- Die vorherige Simulation des Robotereinsatzes kann in realitätsnaher, virtueller Umgebung stattfinden, da sich diese präzise aus dem BIM-Modell erzeugen lässt

Da in immer mehr Bauausschreibungen sowohl privater als auch öffentlicher Auftraggeber die Nutzung von BIM gefordert und durchgesetzt wird, ist auch die effiziente Nutzung von BIM in Kombination mit Robotereinsätzen zukünftig immer realistischer.

2.3. WICHTIGE BEGRIFFE

In diesem Abschnitt werden in dieser Diplomarbeit verwendete Begriffe und Konzepte definiert und kurz erläutert. Die Definitionen erheben dabei keinen Anspruch auf Vollständigkeit oder absolute Wahrheit, da abhängig von Kontext, Nutzer und Sprachraum teilweise mehrdeutige Interpretationen bestehen.

2.3.1. BUILDING INFORMATION MODELING

Die Abkürzung BIM steht für *Building Information Modeling*, also dem Modellieren von Bauwerksdaten. Da der Begriff je nach Anwendung und Kontext verschieden definiert wird und Beteiligte am Baugeschehen teilweise stark abweichende Vorstellung von BIM haben, wird hier die in Deutschland geläufige Definition nach VDI 2552:2020-07 sowie Scherer und Schapke genutzt.

Der Verein Deutscher Ingenieure e.V. sieht BIM als einen *„Oberbegriff für die Digitalisierung in der Wertschöpfungskette [beim] Planen, Bauen und Betreiben [von Gebäuden]“*. Es ist damit gleichzeitig eine Methodik zur Zusammenarbeit auf Grundlage digitaler Bauwerksmodelle. Diese Modelle enthalten für den Lebenszyklus des Bauwerkes relevante Informationen und Daten, welche zwischen allen am Bauwerk beteiligten Parteien ausgetauscht werden können. Um dies zu gewährleisten, sind die Informationen und deren Austausch konsistent und standardisiert. (VDI 2552:2020-07, S. 5)

Scherer et al. bezeichnen BIM als ganzheitliche, *„digitale Betrachtung aller Prozesse und Informationen in einem Bauprojekt“*. Damit wird die Zusammenarbeit zwischen allen am Bauwerk beteiligten Akteuren effizienter und fehlerfreier gestaltet (Scherer & Schapke, 2014, S. 3–7). Beiden Definitionen stellen dabei Digitalisierung und Zusammenarbeit über die gesamte Lebensdauer eines Bauwerkes in den Vordergrund.

Um die Ziele von BIM umsetzen zu können, sind in der VDI-Richtlinie VDI 2552:2020-07 vier sogenannte *Leistungsniveaus* definiert (VDI 2552:2020-07, S. 6–8).

- Das Leistungsniveau 0 bezeichnet die einfache Kooperation zwischen einzelnen Akteuren auf Basis von gedruckten oder digitalen zweidimensionalen Zeichnungen (CAD), einfachen Datenbanken oder Dateienaustausch via Post oder E-Mail.

- Im Leistungsniveau 1 wird der Informationsaustausch überwiegend über eine gemeinsam genutzte Datenumgebung realisiert, ist jedoch nicht standardisiert. Basis sind zwei- oder dreidimensionale CAD-Planungen, die mit Informationen strukturiert verknüpft werden können.
- Das Leistungsniveau 2 erweitert Leistungsniveau 1 um die Nutzung von offenen Dateiformaten und spezifischer „BIM-Autorensoftware“ anstatt bloßer CAD-Software. Informationen und Zeichnungselemente können strukturiert verknüpft und ausgewertet werden. Außerdem wird der Informationsaustausch überwiegend über eine gemeinsam genutzte Datenumgebung realisiert.
- Das Leistungsniveau 3 stellt aktuell das höchste Leistungsniveau dar und erfordert eine vollständig offene Prozess- und Dateiintegration in einer gemeinsam genutzten Plattform. Weitere Niveaus sind denkbar, sind aber derzeit nicht explizit definiert.

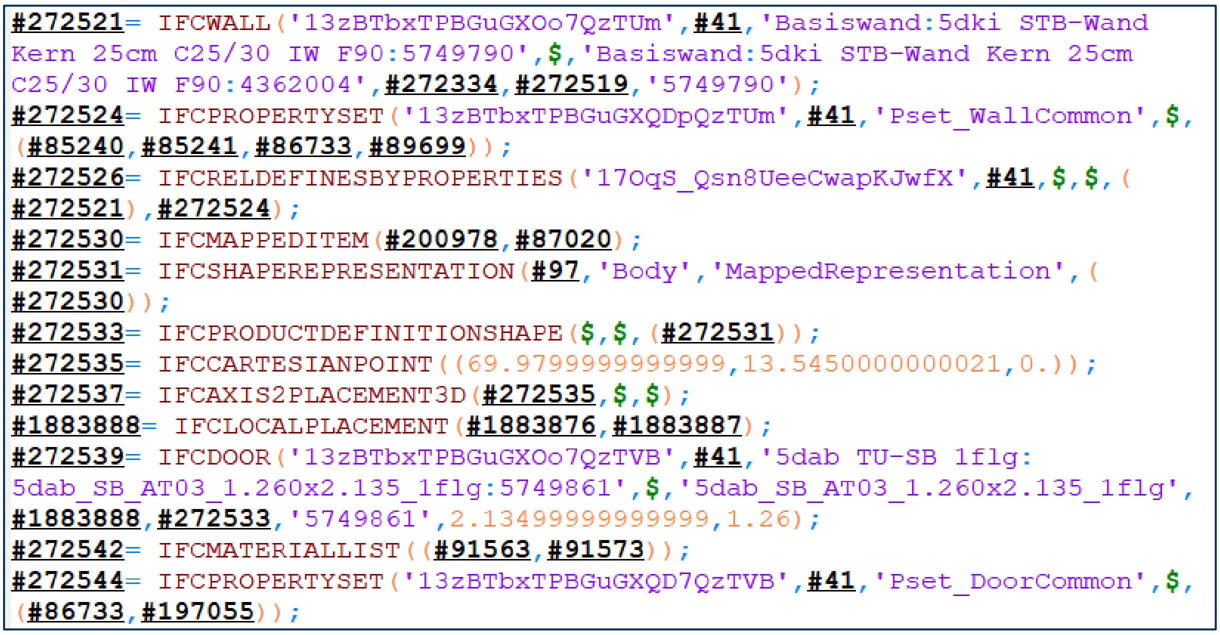
Die Organisation *buildingSMART* hat sich zum Ziel gesetzt, die Vorteile des Building Information Modeling durch Verbesserung der Zugänglichkeit und Zukunftssicherheit zu erweitern. Wichtigste Grundlage dafür ist die Herstellerneutralität, also die Unabhängigkeit von einzelnen Herstellern und deren proprietären Produkten. Daher entwickelt und zertifiziert buildingSMART offene Standards zum Datenaustausch, Arbeitsablauf und Schnittstellen im Rahmen eines BIM-Modells. Ziele des sognannten *openBIM*-Ansatzes sind dabei die effektive Interoperabilität und Zusammenarbeit zwischen den Akteuren im Planungsprozess sowie Bau- und Betriebsprozesses eines Bauwerkes. Weiterhin soll der openBIM-Ansatz durch die offenen, herstellerunabhängigen Standards die Wahl der genutzten Software jedem Akteur freistellen und BIM-Modelle auch zukünftig nutzbar machen, da man nicht auf langfristige Unterstützung einzelner Hersteller und deren proprietäre Produkte angewiesen ist (buildingSMART, 2021b).

Die anhand dieser Ziele entwickelten Standards sind unter anderem das Datenaustauschschema *IFC* (Industry Foundation Classes) und die modellbasierte Kommunikation zwischen BIM-Anwendungen über das Informationsaustauschschema *BCF* (BIM Collaboration Format).

In dieser Diplomarbeit wird mindestens das BIM-Leistungsniveau 2 und die Nutzung des openBIM-Ansatzes vorausgesetzt. Dadurch ist gewährleistet, dass ein ausreichend detailliertes, dreidimensionales Modell des Gebäudes vorliegt, welches durch den offenen Standard auch von verschiedensten Softwarepaketen verarbeitet werden kann.

2.3.2. INDUSTRY FOUNDATION CLASSES

Das von der Organisation *buildingSMART* entwickelte, offene Datenaustauschschema IFC (Industry Foundation Classes) ermöglicht eine standardisierte, digitale Beschreibung von Bauwerken. Dazu gibt es ein Datenschema vor, in dem unter anderem Objekte, deren



```
#272521= IFCWALL('13zBTbxTPBGuGXOo7QzTUm',#41,'Basiswand:5dki STB-Wand
Kern 25cm C25/30 IW F90:5749790',$,'Basiswand:5dki STB-Wand Kern 25cm
C25/30 IW F90:4362004',#272334,#272519,'5749790');
#272524= IFCPROPERTYSET('13zBTbxTPBGuGXQDpQzTUm',#41,'Pset_WallCommon',$,
(#85240,#85241,#86733,#89699));
#272526= IFCRELDEFINESBYPROPERTIES('17OqS_Qsn8UeeCwapKJwfX',#41,$,$,(
#272521),#272524);
#272530= IFCMAPPEDITEM(#200978,#87020);
#272531= IFCSHAPEREPRESENTATION(#97,'Body','MappedRepresentation',(
#272530));
#272533= IFCPRODUCTDEFINITIONSHAPE($,$,(#272531));
#272535= IFCCARTESIANPOINT((69.9799999999999,13.5450000000021,0.));
#272537= IFCAXIS2PLACEMENT3D(#272535,$,$);
#1883888= IFCLOCALPLACEMENT(#1883876,#1883887);
#272539= IFCDOOR('13zBTbxTPBGuGXOo7QzTVB',#41,'5dab TU-SB 1flg:
5dab_SB_AT03_1.260x2.135_1flg:5749861',$,'5dab_SB_AT03_1.260x2.135_1flg',
#1883888,#272533,'5749861',2.13499999999999,1.26);
#272542= IFCMATERIALLIST((#91563,#91573));
#272544= IFCPROPERTYSET('13zBTbxTPBGuGXQD7QzTVB',#41,'Pset_DoorCommon',$,
(#86733,#197055));
```

Abbildung 1 - Screenshot einer IFC-Textdatei im SPF-Format

Beziehungen und Attribute sowie Akteure im gesamten Prozess des Planens, Bauens und Betreibens eines Bauwerkes beschrieben werden können. Ein solches standardisiertes Datenschema ermöglicht es, Informationen zwischen einzelnen Anwendungen im Rahmen von BIM auszutauschen. Durch das offene Format und die Standardisierung ist in einem hohen Maße Informationssicherheit und Fehlerfreiheit gewährleistet, da das Schema jederzeit transparent einsehbar ist (buildingSMART, 2021a). Die detaillierte Standardisierung ist in ISO 16739-1:2018 zu finden. Detaillierte Dokumentationen zu allen IFC-Versionen finden sich außerdem auf der Website von buildingSMART[1].

Durch IFC ist lediglich ein Daten*schema* zum Informationsaustausch definiert. Der konkrete Austausch erfolgt über verschiedene mögliche Datenformate[2] wie beispielsweise SPF (STEP Physical File), XML (Extensible Markup Language) oder JSON (JavaScript Object Notation) und hängt von der gewünschten Nutzung ab (buildingSMART, 2021a).

Am weitesten verbreitet ist dabei das Datenformat SPF, welches in ISO 10303-21:2016 standardisiert ist. Dabei handelt es sich um eine strukturierte Textdatei, in der sowohl Metadaten zur Datei selbst (genutzte Software, Autor etc.) als auch die Informationen zum BIM-Modell enthalten sind. In Abbildung 1 ist ein Ausschnitt einer solchen Textdatei dargestellt[3]. Jede Instanz eines Elementes des IFC-Schemas wird in einer eigenen Zeile definiert und ist eindeutig am Zeilenanfang durch eine vorangestellte Raute, eine positive

---

[1] Siehe standards.buildingsmart.org/IFC/RELEASE/ (Zuletzt abgerufen am 12.07.2021)

[2] Siehe technical.buildingsmart.org/standards/ifc/ifc-formats/ (Zuletzt abgerufen am 12.07.2021)

[3] Die farbigen Hervorhebungen der einzelnen semantischen Elemente wird durch das Textprogramm *Notepad++* realisiert und ist nicht Bestandteil der Datei selbst. Die dafür verwendete UDL (User Defined Language, von Notepad++ verwendete Sprache für Syntax-Highlighting) ist online unter github.com/notepad-plus-plus/userDefinedLanguages/blob/master/udl-list.md zu finden.

ganze Zahl und einem nachfolgenden Gleichheitszeichen nummeriert. Durch diese eindeutige Kennzeichnung kann auf andere Elemente in der Datei referenziert werden. So verweist beispielsweise die Instanz von IfcProductDefinitionShape mit der Nummer #272533 auf die Instanz Nummer #272531 der IfcShapeRepresentation. Innerhalb der runden Klammern werden alle nötigen Attribute des Elementes definiert, wobei nicht genutzte optionale Attribute durch den Platzhalter $ gekennzeichnet sind.

2.3.3. NAVIGATION

Der Begriff *Navigation* leitet sich vom lateinischen *navigare*, also dem *Steuern eines Schiffes* ab. *Steuern* meint hier das sichere Erreichen eines vorher definierten Zielortes. Dazu sind grundsätzlich drei jeweils eng voneinander abhängige Arbeitsschritte nötig:

- Erstellen einer Karte (Umgebungsmodellierung durch ein Navigationsmodell)
- Ortung des Navigierenden in seiner Umgebung (Lokalisierung) und Abgleich von realer Position zu Position auf Karte (Kartenabgleich, sogenanntes map-matching)
- Planung einer Route vom aktuellen Standort zum Ziel

Das Konzept der Navigation ist natürlich nicht auf Schiffe beschränkt und lässt sich analog auf Anwendungen der Robotik anwenden (Haun, 2013, S. 112).

Der Begriff der *Route* ist dabei nicht streng definiert. Umgangssprachlich ist damit meist eine zuvor festgelegte Abfolge aufzusuchender Orte gemeint. Der konkrete *Weg* oder *Pfad* zwischen diesen Orten ist damit aber nicht zwangsläufig bestimmt. Bei einer sehr kleinschrittigen Abfolge von Routenpunkten lassen sich die Begriffe Weg und Route kaum noch trennen. Bei modernen GPS-Geräten mit integrierten Karten wird beispielsweise aus Routenpunkten meist während der Verwendung des Gerätes ein konkreter Weg ermittelt, der je nach aktueller Position immer wieder angepasst wird.

In der Graphentheorie ist außerdem die Unterscheidung zwischen Weg und Pfad nicht eindeutig, da der englische Begriff *path* sowohl als Weg als auch als Pfad übersetzt werden kann. Turau (2009) definiert einen Weg als eine Abfolge von Knoten (Routenpunkten) eines Graphen, bei der keine Verbindung zwischen zwei Knoten mehr als einmal verwendet wird. Dementsprechend werden dort beide Begriffe synonym verwendet, ansonsten wird von einem *Kantenzug* gesprochen. Wird zusätzlich jeder Knoten nur einmal genutzt, wird der Weg als *einfacher Weg* bezeichnet (Turau, 2009, S. 22–23). Auch Tittmann folgt dieser Definition (Tittmann, 2019, S. 15). Steger nutzt hingegen die Begriffe Weg und Kantenzug synonym für eine Abfolge verbundener Knoten. Der Begriff Pfad bezeichnet dort einen Weg, bei dem Kanten nicht mehr als einmal genutzt werden (Steger, 2007, S. 61).

Auch in Veröffentlichungen zum Thema Navigation, Routenplanung und Lokalisierung von mobilen Robotern werden diese Begrifflichkeiten weder im Deutschen noch im Englischen einheitlich verwendet. Damit lassen sich die Begrifflichkeiten aus der

Graphentheorie nur bedingt auf die praktische Umsetzung für mobile Roboter anwenden. Daher wird in dieser Diplomarbeit folgende Festlegung der Begriffe getroffen:

- *Route* – geplante Reihenfolge aufzusuchender Orte (Routenpunkte)
- *Weg* – für das Erreichen der Routenpunkte nötige, konkrete Positionen in der Umgebung des mobilen Roboters
- *Tatsächlicher Weg* – Tatsächlich vom Roboter genutzter Weg (beispielsweise aus von außen getätigten Beobachtungen oder Messungen gewonnene Information)

2.3.4. TRAJEKTORIE

Die in Abschnitt 2.3.3 betrachteten Begriffe zur Navigation befassen sich nur mit dem Aufenthaltsort des Roboters. Die Art und Weise, *wie* der Zielort erreicht werden soll, ist damit nicht definiert. Daher wird zusammenfassend für sämtliche Bewegungsabläufe des Roboters der Begriff *Trajektorie* genutzt. Diese enthält Position, Geschwindigkeit und Beschleunigung aller Antriebe und Gelenke des Robotersystems. Gleichzeitig kann eine Trajektorie aber auch das gesamte Robotersystem als Ganzes betrachten und entsprechend beispielsweise durch Geschwindigkeitsbegrenzungen auf der Baustelle oder maximale Gesamtbeschleunigung aus Sicherheitsgründen eingeschränkt werden (Siciliano & Khatib, 2016, S. 183–187).

Da in dieser Diplomarbeit der Fokus vor allem auf der geometrischen und semantischen Wegfindung und Lokalisierung liegt, wird das Ermitteln und Nutzen von Trajektorien hier nicht weiter vertieft. Randbedingungen aus minimalen und maximalen Geschwindigkeiten sowie deren erste und zweite Ableitung (Beschleunigung und Ruck[4]) ergeben sich vor allem aus der technischen Umsetzung und dem konkret verwendeten Robotermodell im baupraktischen Einsatz.

2.3.5. RAUMBEGRIFFE

Bei der Bezeichnung von Räumen im weitesten Sinne liegen ebenfalls uneinheitliche Definitionen vor. Im Englischen wird in Bezug auf Bauwerke meist der Begriff *spatial structure* vom Begriff *room* oder *space* abgegrenzt. Dies führt bei Übersetzungen ins Deutsche zu Problemen, da alle drei Begriffe als *Raum* beziehungsweise *den Raum betreffend* oder *räumlich* übersetzt werden können.

Spatial structure meint hier eine rein fiktive Begrenzung eines Gebietes als eine Sinneinheit. Das kann beispielsweise eine gesamte Etage eines Gebäudes, der gesamte Parkplatz

---

[4] Der Ruck (im Englischen *jerk*, *jolt* oder *surge* genannt) bezeichnet die Stärke der Beschleunigungsänderung. Er ist beispielsweise als Kraftänderung fühlbar, wenn in einem Fahrzeug das Gaspedal schlagartig stärker durchgetreten wird. Die Einheit des Rucks wird in Weg je Zeiteinheit zur dritten Potenz angegeben, beispielsweise in m/s^3 oder $°/ms^3$.

eines Kaufhauses, eine einzelne Parkbucht auf einem Parkplatz, ein einzelnes Zimmer in einer Wohnung oder der Essbereich in einem Wohnzimmer sein. Aber auch Wände, Türen oder Fahrstuhlschächte sind als *spatial structure* beschreibbar. In dieser Arbeit soll daher allgemein vom *Bezugsraum* gesprochen werden, um Verwechslungen mit dem Wort *Raum* zu vermeiden. Bezugsräume können sowohl mehrere weitere Bezugsräume enthalten als auch selbst in einem anderen Bezugsraum enthalten sein.

Demgegenüber stehen tatsächliche physikalische *rooms* (Räume), welche beispielsweise Zimmer, Büros oder Flure sein können. Räume können ebenfalls mehrere Bezugsräume enthalten oder selbst in einem Bezugsraum enthalten sein.

Die Fläche oder das Volumen eines Raumes oder eines Bezugsraumes wird *space* genannt und teilweise synonym zum Begriff *room* verwendet.

Nach Taneja et al. lassen sich Bezugsräume in folgende vier Kategorien einordnen:

- Freiraum (free space)
- Durchgang (portal)
- Blockade (obstruction)
- Pfad (path)

Diese Einteilung lässt eine Abstraktion der realen, physischen Umgebung für geometrische und topologische Betrachtungen zu. Dafür werden zum Beispiel Wände und Hindernisse allgemein als Blockaden betrachtet; Türen oder durchquerbare Wanddurchbrüche dagegen als Durchgänge zwischen Freiräumen. Somit lassen sich grundlegende physikalische Einschränkungen einfach modellieren: Ein Roboter muss sich immer auf einer Freifläche befinden, kann nicht durch Blockaden hindurch und kann zwischen Freiflächen nur über Durchgänge wechseln (Taneja et al., 2016, S. 25).

2.4. ROS – ROBOT OPERATING SYSTEM

Die Steuerung eines Roboters, egal ob stationär oder mobil, erfordert in der Regel eine gleichzeitige Verarbeitung einer Vielzahl von Prozessen, Informationen und Befehlen. Selbst ein einfacher stationärer Roboterarm muss mehrere Aktionen gleichzeitig ausführen können. Das umfasst beispielsweise die Ansteuerung der einzelnen Motoren der Gelenke, das Überwachen der Umgebung beziehungsweise des Arbeitsraumes, Hinderniserkennung zur Kollisionsvermeidung und Überwachung von Systemparametern wie Stromzufuhr oder aktueller Krafteinwirkung.

Weiterhin spielt immer mehr das Zusammenwirken mehrerer privater und kommerzieller Nutzer, Hersteller und Entwickler an einem Projekt eine Rolle. Komplexere Robotersysteme bauen auf verschiedenste Systemkomponenten auf, die von verschiedenen Herstellern geliefert werden. Daher sind standardisierte Schnittstellen und ganzheitliche

Entwicklungsumgebungen von großem Nutzen. Eine solche Umgebung bietet das Software-Framework *Robot Operating System*, kurz ROS (Open Robotics, 2020b, 2020c). Nachdem es 2007 vom *Stanford Artificial Intelligence Laboratory* im Rahmen des Projektes *Stanford AI Robot* (STAIR) ins Leben gerufen wurde, übernahm ab 2008 das Institut *Willow Garage* dessen Entwicklung. Seit 2012 erfolgt die Weiterentwicklung gemeinsam mit der gemeinnützigen *Open Source Robotics Foundation* (OSRF) und seit 2013 allein durch die OSRF[5] (Quigley et al., 2015; Ravichandran, 2013).

ROS ist in seinem Aufbau vergleichbar mit einem Betriebssystem und wird zum Entwickeln sowie Ausführen und Simulieren von Software für Roboter genutzt. Dafür bietet es zahlreiche Hilfswerkzeuge an und unterstützt parallele Berechnungen über mehrere Computersysteme hinweg. Durch den konsequent modularen Aufbau ist es robust gegenüber Systemänderungen während der Entwicklungszeit und unterstützt Schnittstellen zwischen zahlreichen Komponenten. Ein weiterer Vorteil ist die relative freie Wählbarkeit unterschiedlicher Programmiersprachen. So werden unter anderem C++, Python und Java unterstützt.

Da ROS mit einer freizügigen Lizenz der *Berkeley Software Distribution* (kurz als BSD bezeichnet) als freie Open-Source-Software[6] zur Verfügung steht, gut dokumentiert ist und eine große, aktive und hilfsbereite Entwicklercommunity hat, wird ROS auch in dieser Diplomarbeit genutzt. Zusätzlich wird dadurch weitgehende Unabhängigkeit von proprietären Produkten einzelner Hersteller erreicht. Einzelne Funktionalitäten lassen sich als Module anderer Entwickler ergänzen und Kombinieren (Quigley et al., 2015; Ravichandran, 2013).

2.4.1. VERSIONEN UND DISTRIBUTIONEN

Das Robot Operating System wird derzeit in zwei Hauptversionen angeboten: ROS und ROS 2. Die aktuelle und voraussichtlich letzte Version von ROS 1 ist *ROS Noetic*, die bis Mai 2025 weiterentwickelt werden soll. ROS wird offiziell ohne die Versionsnummer „1“ bezeichnet. Zur eindeutigen Abgrenzung zu Version ROS 2 wird in dieser Diplomarbeit aber immer die Bezeichnung ROS 1 verwendet, wenn sich Informationen ausschließlich auf ROS 1 beziehen (Open Robotics, 2021a).

Die erste stabile Version von ROS 2 wurde im Dezember 2017 veröffentlicht. Die neuste Version von ROS 2 ist *Galactic Geochelone*, welche allerdings nur bis November 2022 unterstützt wird. Die aktuelle Version mit Langzeitsupport (LTS) ist *Foxy Fitzroy*, welche am 05. Juni 2020 veröffentlicht wurde und bis Mai 2023 weiterentwickelt wird (Open Robotics, 2021b).

---

[5] Website der OSRF unter openrobotics.org erreichbar (Zuletzt abgerufen am 21.07.2021)

[6] Der Quellcode von ROS ist online als komplettes Repository verfügbar auf der Plattform GitHub unter github.com/osrf (Zuletzt abgerufen am 21.07.2021).

Zwischen den Hauptversionen ROS 1 und ROS 2 bestehen mehrere fundamentale Unterschiede, welche vor allem die Unterstützung von Echtzeitberechnungen, mehreren Robotern, neuere Programmiersprachen und Schnittstellen innerhalb von ROS-Komponenten betreffen. In Tabelle 1 sind die bedeutendsten Unterschiede aufgelistet (Open Robotics, 2019; Thomas & Open Robotics, 2017a, 2017b).

Tabelle 1 - Unterschiede zwischen ROS 1 und ROS 2

Bereich	**ROS 1**	**ROS 2**
Betriebssystem	Nur unter Linux Ubuntu getestet, andere nur inoffiziell	Offizielle Unterstützung von Linux Ubuntu, OS X, macOS und Windows 10 sowie teilweise Arch Linux und andere
Genutzte Programmiersprachen	Python 2 C++ Version 03	Python 3.5 oder höher C++ Version 11, 14
Middleware (Kommunikation zwischen Prozessen und Komponenten)	Transportprotokoll und Kommunikation eigenständig für ROS 1 als *roscore* entwickelt: • Nicht echtzeitfähig • Mehrere Roboter nur sehr eingeschränkt unterstützt	Nutzung des offenen Standards *Data Distribution Service*[7], wodurch sich folgende Vorteile ergeben: • Echtzeitfähig • Mehrere Roboter zugleich nutzbar • Deutlich verbesserte Kommunikationsqualität zwischen Komponenten
Erstellen von Paketen	Nur mit ROS-spezifischen Build-Tools erstellte Pakete sind nutzbar	Unterteilung in Pakete und Arbeitsumgebung, wodurch die Pakete nicht mehr von ROS-spezifischen Build-Tools abhängen
Programmstart	Nutzung des Hilfsprogramms *roslaunch* mithilfe von XML-Dateien	Programmstart als Python-Skript beschreibbar, was komplexere Startsequenzen ermöglicht

Die verschiedenen Unterversionen von ROS 1 und 2 werden als *Distributionen* bezeichnet. Eine Distribution ist hierbei eine Zusammenstellung einzelner Softwarekomponenten. Im Fall von ROS sind dies – ähnlich einem regulären Betriebssystem – beispielsweise

---

[7] Standard online verfügbar bei der *Object Management Group* unter omg.org/spec/DDS/ (Zuletzt abgerufen am 21.07.2021)

Kommunikationsroutinen (bei ROS: *Messages*), Programme (*Nodes*) und Programmierumgebungen (*Software-Frameworks*) sowie Skripte und Dokumentationen (Ravichandran, 2013). Die konkrete Wahl der Distribution hängt vor allem davon ab, welche Pakete man benötigt und ob diese in der Distribution verfügbar und kompatibel sind sowie der gewünschten Supportdauer. In dieser Diplomarbeit wird die Distribution *Foxy Fitzroy* von ROS 2 genutzt, da diese mit dem Fokus auf Stabilität und Kompatibilität als LTS-Version bis 2023 unterstützt und weiterentwickelt wird. Die Wahl von ROS 2 anstatt ROS 1 ergab sich außerdem durch die Nutzung von Python 3 anstatt der veralteten Python-Version 2 sowie die Verfügbarkeit des Paketes *nav2*, welches leistungsfähige Komponenten zur Lokalisation und Navigation für Roboter in bekannten und unbekannten Umgebungen bietet.

	ROS Level	
	Level Content	**Level components**
Community Level	• Distributions • Repositories • ROS Wiki • Other resources	• Web access
Computational Level	• Master • Nodes • Messages • Topics • Services • Bags	• Middleware ○ Messaging ○ Libraries and Compiler ○ Software Management • Simulators • Visualizers • Debugger • Package Management ○ Robot Control ○ Motion planning ○ Localization and Mapping ○ Computer Vision ○ Processing sensor information ○ etc.
File System Level	• Packages • Stacks • Manifests • Message types • Service types	

Abbildung 2 - Ebenen des Robot Operating System

2.4.2. AUFBAU UND VERWENDUNG

Die gesamte Architektur des *Robot Operating System* basiert auf vier grundlegenden Annahmen. Die erste Annahme ist, dass Roboter immer *Aufgaben* erfüllen. Eine Aufgabe kann dabei erst einmal beliebig abstrakt formuliert sein. Ein einfacher Roboterarm könnte beispielsweise die Aufgabe „Nimm ein Bauteil und lege es auf den Werktisch" erfüllen. Eine Aufgabe im Kontext dieser Diplomarbeit könnte zum Beispiel „Fahre von der Ladestation zum aktuellen Arbeitsplatz" lauten. Eine solche abstrakte Formulierung ist daher üblicherweise ein komplexes Gebilde aus vielen Unteraufgaben, welche wiederrum selbst in weitere Unteraufgaben zerlegbar sind. Dies stellt die zweite Annahme dar. Die zuvor formulierte Aufgabe des Roboterarms ließe sich beispielsweise in die Komponenten *Pfadplanung*, *Gelenksteuerung*, *Steuerung des Greifers*, *Objekterkennung aus Kamerabildern* und *Kollisionserkennung mithilfe von Näherungssensoren* untergliedern. Durch ausreichende Abstraktion der jeweiligen Aufgaben lassen sich diese auch in anderen Anwendungen wiederverwenden. Für die Objekterkennung mithilfe einer Kamera ist es beispielsweise irrelevant, ob diese an einem Roboter mit zwei oder vier Gelenken genutzt wird. Daher ist die dritte Annahme, dass Aufgaben so weit wie möglich abstrahiert sind, um deren einfache Wiederverwendung in anderen Aufgaben zu ermöglichen. In ROS wird eine ausreichend abstrahierte Unteraufgabe als *Komponente* bezeichnet.

Ebenen

Die Gesamtheit des Robot Operating System lässt sich auf drei Ebenen beziehungsweise *Level* einteilen und betrachten: *File System Level*, *Computational Level* und *Community Level*. In Abbildung 2 sind die drei Ebenen mit ihren jeweiligen Inhalten und Komponenten aufgelistet. Das Community Level ist dabei am weitesten gefasst und beschreibt sämtliche Wissenssammlungen, Dokumentationen und Sammlungen von Quellcode sowie die Zusammen- und Bereitstellung von ROS-Distributionen. Der Zugriff erfolgt dabei einfach über verschiedene Internetseiten. Die Ebene des Computational Level enthält abstrakte Konzepte für einzelne Komponenten, wie zum Beispiel Schemata für Nachrichten zwischen Programmteilen oder den Aufbau von Anweisungen. Die unterste Ebene stellt das File System Level dar. In ihr werden beispielsweise Datenstrukturen für Komponenten definiert, Metadaten bereitgestellt und Pakete organisiert. Die tatsächliche Umsetzung der Komponenten des Computational Level wird demnach durch das File System Level realisiert (Open Robotics, 2019; Quigley et al., 2015; Ravichandran, 2013).

Komponenten

Die Kommunikation zwischen Komponenten und Prozessen, das Bereitstellen von Programmierbibliotheken und Compilern für verschiedene Programmiersprachen sowie das gesamte Software-Management werden über die sogenannte *Middleware* bereitgestellt. Um Fehler effizient finden zu können, sind in ROS außerdem verschiedene Debugger enthalten, die beispielsweise Programmcode schrittweise ausführen können, Sensordaten simulieren oder ausgeführte Arbeitsschritte speichern und erneut ausführen können, um identische Testbedingungen zu sichern. Weitere wichtige Komponenten in ROS sind Simulatoren, die virtuelle Testräume für die erstellten Algorithmen bereitstellen sowie Visualisierer, die Simulationsergebnisse grafisch darstellen. Die eigentliche Steuerungslogik der Roboter befindet sich in Komponenten, die als *Pakete* beziehungsweise *Packages* bezeichnet werden. Dazu zählen sowohl direkt die Steuerung des Roboters betreffende Komponenten wie Motorsteuerung oder das Aufnehmen von Sensordaten als auch indirekte Komponenten wie Sensorauswertung, Wegplanung oder Bilderkennungssoftware.

Die Nutzung von ROS als reiner Anwender findet dabei fast ausschließlich im Bereich der Paketverwaltung in Kombination mit Debuggern, Simulatoren und Visualisierern statt. Man kombiniert, bearbeitet und erstellt Pakete, die nach eingehenden Tests auch zur Steuerung von realen Robotern genutzt werden können.

ROS Graph
Die zuvor beschriebene Modularisierung in einzelne Komponenten folgt einem konkreten Informationsschema, welches den sogenannten *ROS Graph* ergibt und in Abbildung 3 dargestellt ist. Dieser Graph besteht aus *Nodes* (Knoten), die durch *Edges* (Kanten) verbunden sind. Dabei stellt jeder Knoten eine einzelne Komponente dar, während jede Kante Nachrichten zwischen Knoten repräsentiert. Durch diesen Aufbau ist der Austausch und die Wiederverwendung einzelner Komponenten oder ganzer Teilsysteme – also Subgraphen mit mehreren Knoten – mit geringem Aufwand möglich. Die Kommunikation zwischen den Knoten wird in ROS 1 mithilfe der eigens dafür erstellten Middleware-Komponente *roscore* koordiniert, während in ROS 2 der Standard *Data Distribution Service* zur Umsetzung der Kommunikation über Middleware genutzt wird (siehe dazu auch den Vergleich in Tabelle 1 auf Seite 20).

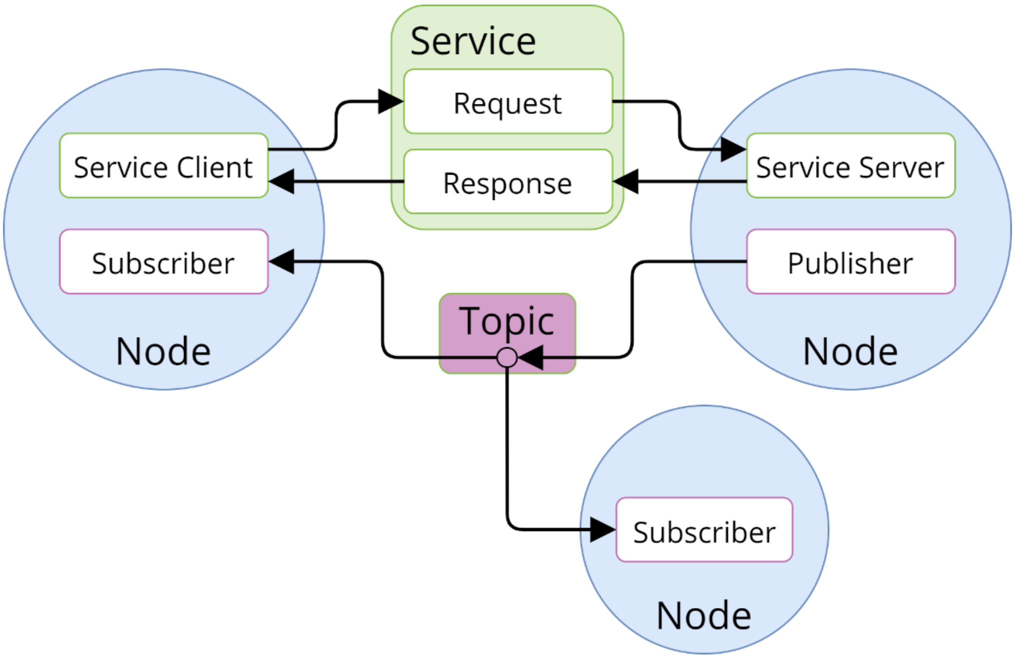


Abbildung 3 - Darstellung der ROS-Komponenten und deren Kommunikation im ROS Graph

Kommunikation im ROS Graph

Informationsaustausch zwischen den Knoten kann auf zwei Arten erfolgen. Die erste, einfache Variante ist ein einseitiges Aussenden und Empfangen von Nachrichten nach dem *Publisher-Subscriber-Prinzip*. Der Publisher sendet thematisch identische Nachrichten dauerhaft als sogenannte *Topics* aus und jeder Knoten, der als Empfänger diese Informationen empfangen möchte, trägt sich als *Subscriber* für diese Topics ein. Eine direkte Rückmeldung des Empfängers auf den Nachrichteninhalt ist hier nicht vorgesehen. Inhalt der Nachrichten können von einfachen Sensordaten wie die aktuelle Spannung des Roboterakkus bis hin zu aus komplexen Berechnungen resultierende Informationen sein. So kann zum Beispiel durch ein Teilsystem aus mehreren Knoten ermittelt werden, ob der gewünschte Fahrtweg frei oder blockiert ist und diese Information anschließend an die Navigationseinheit weitergegeben werden, welche dann eventuell eine Neuberechnung der Route anordnet.

Die zweite Möglichkeit besteht darin, einen *Service* (Dienst) anzufragen, auf den eine Aktion und eine Antwort erfolgt. Im Gegensatz zu den dauerhaft ausgesendeten Topics muss ein Service explizit jedes Mal angefragt werden. Dabei wird das *call-and-response-Modell* verwendet: ein *Client* fragt einen Dienst an (*Request Call*) und erhält eine Antwort (*Response*) vom Dienstanbieter, dem *Server*. Ein Dienst kann dabei von beliebig vielen Klienten angefragt werden. Grundlegende Dienste, die die meisten Knoten in ROS anbieten, sind Dienste zum Abfragen und Setzen von Parametern, sogenannte *Getter*- und *Setter-Dienste*. Durch diese lassen sich während der Programmausführung die Eigenschaften (Parameter) von Knoten abfragen und verändern. Speziellere, praxisnahe Dienste sind dagegen beispielsweise die Anfrage der Ladesteuerung des Roboterakkus an die

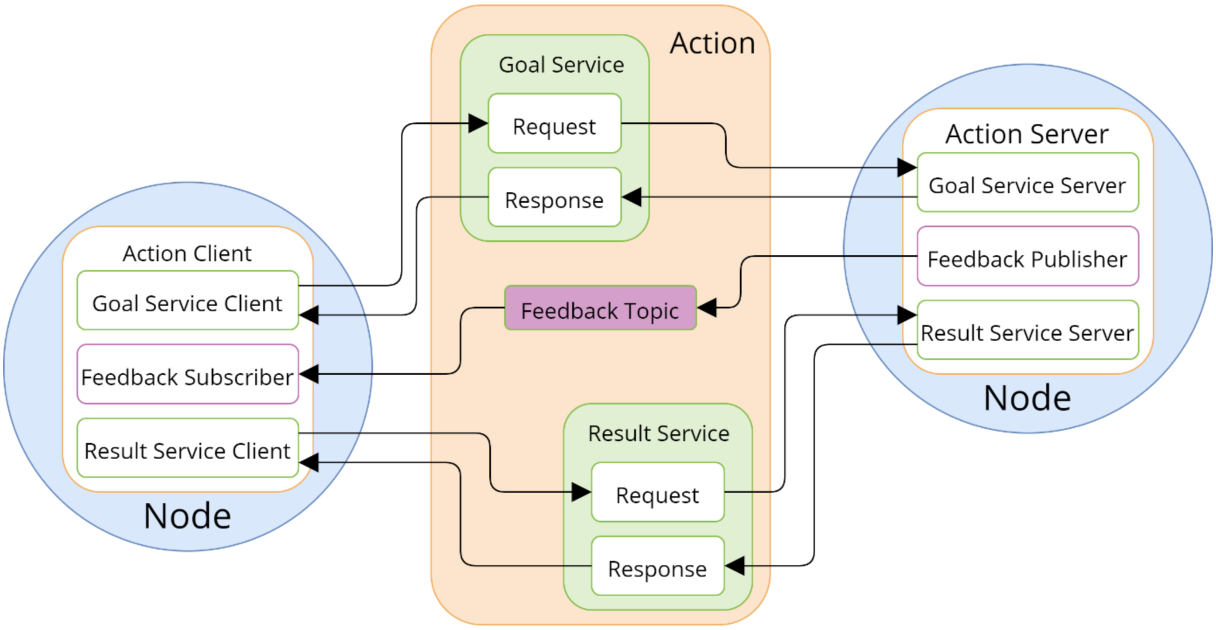


Abbildung 4 - Kombination von Topics und Services zu einer Action in ROS

Navigation, den Weg zur Ladestation zu suchen, nachdem ein niedriger Ladezustand erkannt wurde.

Eine Kombination aus Topics und Services dient zur Ausführung langanhaltender Aufgaben und wird in ROS als *Action* bezeichnet. Im Gegensatz zu reinen Services sind Actions durch den Klienten steuerbar: sie lassen sich abbrechen und liefern dauerhaft Rückmeldungen an den Klienten. In Abbildung 4 ist der Aufbau einer Action schematisch dargestellt (Open Robotics, 2021c; Quigley et al., 2015, S. 7–11).

2.5. GRAPHENTHEORIE

Da in dieser Diplomarbeit immer wieder auf die Wissensrepräsentation durch Graphen zurückgegriffen wird, sind im Folgenden grundlegende Begriffe und Konzepte der Graphentheorie erklärt.

2.5.1. GRUNDBEGRIFFE DER GRAPHENTHEORIE

Ein *Graph* G im Sinne der Graphentheorie besteht aus einer Menge *Knoten* V und einer Menge *Kanten* E. Jeder Kante $e \in E$ sind dabei zwei Knoten $v \in V$ zugeordnet, welche als *Endknoten* der jeweiligen Kante bezeichnet werden. Die Kurzbezeichnungen leiten sich aus den englischen Begriffen *vertex* für Ecke beziehungsweise Knoten und *edge* für Kante ab.

Sind die Kanten durch ungeordnete Paare von Knoten definiert, ist der Graph *ungerichtet*. Sind dagegen die Kanten durch geordnete Paare von Knoten definiert, spricht man von einem *gerichteten* Graph. Gerichtete Kanten haben also einen Start- und einen Endknoten und dementsprechend eine definierte Richtung.

Werden die Kanten oder Knoten zusätzlich durch *Gewichte* bewertet, handelt es sich um einen *kantenbewerteten* beziehungsweise *knotenbewerteten Graph*. Gewichte können beispielsweise Zahlenwerte oder boolesche Variablen (Wahr/Falsch) sein. In Abbildung 5 ist

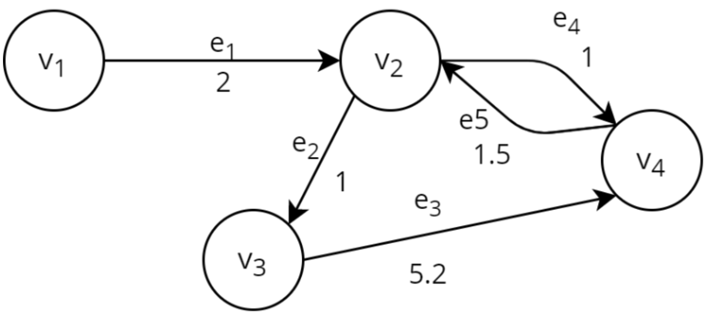


Abbildung 5 - Ein gerichteter, kantengewichteter Graph beispielhaft ein gerichteter Graph dargestellt, der zusätzlich gewichtete Kanten aufweist (Tittmann, 2019; Turau, 2009).

2.5.2. HYPERGRAPHEN

Beim Konzept der Hypergraphen handelt es sich um eine Verallgemeinerung der Graphentheorie. Ein Hypergraph ist demnach ein Graph, dessen Kanten als *Hyperkanten* mehr als nur zwei Knoten verbinden, indem sie diese Umschließen (Bretto, 2013, 23-27).

Im Abschnitt 3.5.2 in Abbildung 13 ist ein einfacher Hypergraph für einen Teil eines Beispielgebäudes dargestellt. Die Knoten entsprechen dabei Durchgängen zwischen Räumen, während die Hyperkanten die Räume selbst beziehungsweise deren Grenzen repräsentieren. Als abstraktere Beispiele für Hypergraphen können auch elektronische Schaltpläne und Venn-Diagramme angesehen werden.

2.5.3. SPEICHERUNG VON GRAPHEN

Die einfachste Form der Speicherung eines Graphen besteht in der Beschreibung als Matrix. Dies hat außerdem den Vorteil, dass Operationen und Methoden der linearen Algebra auf Matrizen und damit indirekt auf den Graphen anwendbar werden. Ein Graph lässt sich grundlegend in einer *Adjazenzmatrix* und in einer *Inzidenzmatrix* darstellen, die nachfolgend erläutert werden (Tittmann, 2019, S. 29).

Adjazenzmatrix
In einer Adjazenzmatrix $A(G)$ werden alle Kanten zwischen den Knoten eines Graphs eingetragen. Aus ihr lässt sich also erkennen, welche Knoten direkt miteinander verbunden sind. Die Größe der Matrix ergibt sich aus der Anzahl der Knoten n_V zu $n_V \times n_V$. Ein Eintrag a_{ij} steht dabei für die Anzahl der Verbindungen zwischen den Knoten v_i und v_j. Alternativ kann statt der Anzahl der Verbindungen auch die Wichtung der Kanten eingetragen werden. Für das Beispiel aus Abbildung 5 lautet die Adjazenzmatrix inklusive der Gewichte dementsprechend:

$$A(G) = \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 5.2 \\ 0 & 1.5 & 0 & 0 \end{bmatrix} \tag{2.1}$$

Inzidenzmatrix
Die Inzidenzmatrix $I(G)$ gibt für alle Kanten j an, welche Knoten i durch sie verbunden werden. Die Größe der Inzidenzmatrix ergibt sich aus der Anzahl der Knoten n_V und der Anzahl der Kanten m_E zu $n_V \times m_E$. Damit repräsentiert jede Spalte eine entsprechend nummerierte Kante. Für gerichtete Kanten gibt es die Möglichkeit, die Richtung durch das Vorzeichen darzustellen. Eingehende Kanten können beispielsweise durch ein negatives

und ausgehende Kanten durch ein positives Vorzeichen gekennzeichnet werden. Für das Beispiel aus Abbildung 5 ergibt sich dafür folgende Inzidenzmatrix:

$$I(G) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 1 & -1 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 1 \end{bmatrix} \tag{2.2}$$

Weiterhin lassen sich einige Graphen als Baumstruktur speichern, wodurch eine Hierarchie repräsentiert werden kann. Auch relationale Datenbanken können als Graphen aufgefasst beziehungsweise Graphen als relationale Datenbank gespeichert werden. Für eine effiziente Verarbeitung von Graphen existieren weiterhin optimierte, binäre Speicherformen, die individuell an die genutzte Software wie beispielsweise *Neo4J* oder *Protegé* gebunden sind.

2.5.4. WEGFINDUNG IN GRAPHEN

Um nach bestimmten Kriterien optimierte Wege zwischen Knoten eines Graphen zu finden, existiert eine Vielzahl Algorithmen. Der im Jahr 1956 von Edsger W. Dijkstra entwickelte *Dijkstra-Algorithmus* spielt hierbei eine besondere Rolle, da dieser *immer* den kürzesten Weg zwischen zwei Knoten ermitteln kann. Die mathematische Beweisführung, dass dieser Algorithmus immer den kürzesten Weg findet, ist beispielsweise Turau (2009, S. 259–261) zu entnehmen. Seit dessen Veröffentlichung wurden viele Variationen und Optimierungen des Dijkstra-Algorithmus entwickelt, um unter bestimmten Umständen schnellere Suchergebnisse zu erhalten.

Hierbei spielt der *A*-Algorithmus* eine besondere Rolle, da dieser unter Angabe von Start- und Zielknoten basierend auf Heuristik die Komplexität und damit die Laufzeit des Problems von $\mathrm{O}(\mathrm{V}^2)$ auf $O(E)$ verringert. Das bedeutet, dass durch vorherige Schätzung des Ergebnisses anstatt einer quadratischen Laufzeit-Abhängigkeit von der Anzahl Knoten V nur eine lineare Abhängigkeit von der Anzahl der Kantenanzahl E besteht. Die Schätzung des Abstandes eines Knoten i zu einem Zielknoten z erfolgt durch eine Schätzfunktion $f(i)$ dermaßen, dass die geschätzte Distanz immer kleiner oder gleich der tatsächlichen Distanz ist. Es gilt also immer

$$0 \leq f(i) \leq d(i, z) \tag{2.3}$$

Setzt man die Schätzfunktion zu Null, ergibt sich der Dijkstra-Algorithmus. Demnach lässt sich A* auch als Verallgemeinerung des Dijkstra-Algorithmus ansehen (Turau, 2009).

3 POSITIONIERUNG UND NAVIGATION IN GEBÄUDEN

In diesem Kapitel werden Möglichkeiten der Lokalisierung und deren technische Umsetzung durch Sensoren erläutert. Weiterhin werden Navigationsmodelle vorgestellt, die das map-matching, also das Abgleichen von realer Position mit der Position auf einer Karte, sicherstellen.

Da potentielle Einsatzorte eines Roboters vor allem größere Gebäude mit vielen einzelnen Räumen sind, kann dieser sich aus Zeit- und Ressourcengründen nicht ausschließlich auf die eigenständige Kartierung der Umgebung stützen. Dies würde viel zu lange dauern und selbst für ein vollständig vom Roboter kartiertes Gebäude fehlen sämtliche semantische Informationen wie Raumnutzung oder Materialinformationen. Die Arbeitsumgebung muss auf höherer Ebene sowohl semantisch, topologisch als auch geometrisch bekannt sein, gleichzeitig soll der Roboter aber auf temporäre, nicht detailliert planbare Situationen wie blockierte Durchgänge oder sich im Weg befindliche Hindernisse autonom reagieren können. Zusätzlich sind im Ergebnis der praktischen Bauausführung immer Abweichungen vom geplanten Modell vorhanden, welche durch falsche Modellierung, Planungsfehler, ungenaue Ausführung oder Messfehler entstehen. Daher sollen Wegplanung und Kartenerstellung eine Kombination aus dem Wissen des BIM-Modells und den eigenen Erkenntnissen des Roboters sein. Dieses Vorgehen wird durch Nutzung von *Adaptive Monte Carlo Localization* (AMCL), also dem Lokalisieren in einer Karte aufgrund von Umgebungsbeobachtungen sowie dem *Simultaneous Localization and Mapping* (SLAM), also dem gleichzeitigen Lokalisieren und Kartieren, ermöglicht.

3.1. ÜBERBLICK UMGEBUNGSSENSORIK

Um einen Roboter sinnvoll nutzen zu können, benötigt er Sensoren. Diese messen physikalische Eigenschaften, welche in engem, sinnvollen Zusammenhang mit der unmittelbaren Umgebung des Roboters stehen. Sie erfüllen nach Haun (2013, S. 222–223) dabei folgende Aufgaben:

- Erfassen der inneren Zustände des Roboters wie Lage, Geschwindigkeit, Beschleunigung oder Krafteinwirkungen
- Erfassen der Umgebung wie Abstände, Formen oder Oberflächen
- Messen physikalischer Größen wie Temperatur, Feuchte oder Luftdruck
- Erfassen von Werkstücken, Arbeitsorten und deren Wechselwirkungen

- Allgemeine Analyse von Situationen und Szenen der Umwelt

Da es in dieser Diplomarbeit vor allem um die Lokalisierung und Wegfindung innerhalb von Gebäuden geht, werden dafür nicht relevante Sensoren vor allem auf unterster Betriebsebene wie Batteriesensoren, Messung der Gelenkstellung oder Motorströme nicht weiter beachtet. Die folgende Liste nach Haun (2013, S. 223) ist eine Auswahl von möglicherweise für die Erfüllung der gestellten Aufgabe nötigen Sensorkategorien:

- Interne Sensoren
    - Position und Orientierung des Roboters
    - Krafteinwirkungen auf Gelenke und Fahrwerk
- Externe Sensoren
    - Kollisionserkennung
    - Entfernungsmessung
    - Lage von Markierungen und Objekten
    - Umrisse oder Konturen von Objekten
    - Bilder der Umwelt (Licht, Wärme, Schall usw.)
- Oberflächensensoren
    - Tastsensoren

Externe Sensoren lassen sich nach Haun (2013, Abbildung 5.1) und Siciliano und Khatib (2016) weiterhin anhand ihrer Nutzung unterteilen. Diese Nutzungskategorien können sich allerdings auch teilweise überschneiden und lassen sich nicht scharf trennen. Die Einteilung in Tabelle 2 ist daher für den Zweck eines mobilen Roboters auf einer Baustelle angepasst. Die Sensoren werden in Abschnitt 3.2 näher erklärt.

Tabelle 2 - Nutzungskategorien externer Sensoren

Nutzungskategorie	**Sensorbeispiele**
Taktile Sensoren	Tasten, Greifen, Kraft-Moment-Sensoren
Näherungssensoren	Induktion, Kapazitativ, Optisch, Akustisch
Abstandssensoren	Optisch, RADAR, SONAR, LIDAR, Akustisch
Positionssensoren	GNSS, Bluetooth, WLAN, Landmarken
Visuelle Sensoren	Photodioden, CCD (digitale Pixelbilder)

In einem umfangreichen Literaturanalyse untersuchten Karimi und Iordanova (2021, S. 15) in anderen Projekten tatsächlich umgesetzte Möglichkeiten der Lokalisierung von Robotern auf Baustellen. Analog zu den zuvor genannten Nutzungskategorien arbeiteten die AutorInnen Lokalisierungssysteme nach der Grundlage der genutzten Messgröße heraus. In Tabelle 3 ist eine Zusammenfassung der Einteilung dargestellt. Anhand der Analyseergebnisse lässt sich feststellen, dass vor allem eine Kombination wellenbasierter

Techniken – insbesondere LIDAR und Infrarot – mit Bildaufnahmen durch Digitalkameras erfolgsversprechend sind.

Aufgrund der breiten Anwendbarkeit wellenbasierter Entfernungsmessungen werden in Abschnitt 3.2 deren Grundlagen und Anwendungen erläutert, um potentielle Nutzungsmöglichkeiten sowie Vor- und Nachteile im Rahmen dieser Diplomarbeit bewerten zu können.

Tabelle 3 - Grundlegende Kategorie der Messgrößen

Grundlage	**Umsetzungsbeispiele**
Wellen	LIDAR, RADAR, IR, Ultraschall etc.
Bilder	Digitalkameras
Bewegungsmessung	Zählen der Radumdrehungen, Drehwinkelbestimmung von Gelenken etc.

3.2. ENTFERNUNGSMESSUNG

Sensoren zur Messung von Entfernungen bilden die dreidimensionale Oberflächenstruktur der Umgebung ab. Dazu wird meist die direkte Entfernung vom Sensor zur abzutastenden Oberfläche ermittelt, oft auch als Tiefenmessung beziehungsweise *depth measuring* bezeichnet. Aus den daraus resultierenden Repräsentationen der Entfernungsdaten müssen anschließend verwertbare Merkmale extrahiert werden. Dazu zählen beispielsweise Ebenen, Linien und Oberflächen (Konolige & Nüchter, 2016, 783).

3.2.1. REPRÄSENTATION DER TIEFENDATEN

Die aktuell abgetastete Umgebung wird *Szenerie* (englisch S*cene*) genannt. Abtasten bedeutet hierbei, dass die Koordinaten (x, y, z) mindestens eines sichtbaren Oberflächenpunktes im Koordinatensystem (X, Y, Z) ermittelt werden. Da nur die sichtbaren Oberflächen abgetastet werden können, wird in diesem Zusammenhang auch von 2.5-D-Repräsentationen gesprochen. Erst durch Messungen aus verschiedenen Richtungen lässt sich ein echtes 3D-Abbild der Umgebung generieren. Jeder abgetastete Punkt wird in eine Liste von 3D-Datenpunkten $\{(x_i, y_i, z_i)\}$ gespeichert. Eine Menge dieser Punkte nennt man Punktwolke. Aus dieser lässt sich anschließend eine zweidimensionale Repräsentation der Scene erzeugen. Dazu wird jedem Punkt der Abstand Punkt zu Sensor $d(i, j)$ sowie ein Bildpixel (i, j) zugeordnet. Die Abbildungsvorschrift zwischen $\left(i, j, d(i, j)\right)$ und (X, Y, Z) hängt dabei von der Geometrie des Sensors und der gewünschten Anwendung des Ergebnisses ab.

Entfernungsmessgeräte mit rotierenden Spiegeln nutzen beispielsweise meist ein sphärisches Koordinatensystem (θ, φ, d). In seiner einfachsten Form gilt:

$$\begin{gathered} i = \theta \\ j = \varphi \end{gathered} \tag{3.1}$$

wodurch sich die Abbildungsvorschrift zu

$$d = \sqrt{x^2 + y^2 + z^2} \tag{3.2}$$

$$\theta = \arccos\left(\frac{z}{r}\right) \tag{3.3}$$

$$\varphi = \mathrm{arctan2}\ (y, x) \tag{3.4}$$

ergibt. Durch die Projektion einer Kugeloberfläche in den ebenen zweidimensionalen Raum kommt es zu Verzerrungen des Abbildes. Weitere übliche Projektionsmethoden sind unter anderem die *Gnomonische Projektion*, bei der das Projektionszentrum im Zentrum im Mittelpunkt der Scene liegt sowie Mercator-Projektion, Pannini-Projektion oder Zylindrische Projektion (Konolige & Nüchter, 2016, S. 783–785).

3.2.2. ERMITTELN DER TIEFENDATEN

Zur Entfernungsmessung stehen zwei grundlegende Methoden mit jeweils einer Vielzahl von Abwandlungen zur Verfügung. Während sich die *Triangulation* nur auf trigonometrische Berechnungen stützt, wird für *Laufzeitmessungen* (sogenannte TOF, Time-of-Flight-Messungen) die Zeit zwischen Senden und Empfangen eines Messsignals ausgenutzt.

Bei einer Entfernungsmessung durch *Triangulation* wird der Winkel im Dreieck zwischen dem zu messenden Punkt sowie zweier Sensoren gemessen. Der mögliche Messbereich der Tiefe vor den Sensoren wird als *Sichtfeld* oder FOV (Field of View) bezeichnet. Die Genauigkeit der ermittelten Entfernung sinkt dabei quadratisch mit dem Abstand des Punktes zum Sensor. So ergeben sich beispielsweise bei einer ermittelten Abweichung von 1 mm auf 1 m Abstand zwischen Sensor und Punkt bereits 4 mm Abweichung bei einem Abstand von 2 m. Die Genauigkeit lässt sich umgekehrt proportional durch Erhöhen des Sensorabstandes oder Verkleinern des FOV verbessern.

Im praktischen Einsatz werden zur Triangulation entweder Stereokameras oder eine einzelne Kamera in Kombination mit einem Projektor verwendet. Stereokameras nehmen ein Bild der Szenerie aus leicht unterschiedlichen Richtungen auf. Durch Mustererkennung werden gleiche Objekte in beiden Bildern erkannt und anhand dessen die Verschiebung und damit der Abstand zum Objekt ermittelt. Problematisch ist dabei, dass im

Innenbereich selten gut erkennbare Muster existieren. Auf einer glatten, weißen Wand kann beispielsweise kaum ein Muster gefunden werden. Daher wird oft auf eine Einzelkamera in Kombination mit einem Projektor zurückgegriffen. Dieser projiziert ein helles Muster auf die abzutastende Oberfläche, welches durch die Kamera beobachtet wird. Durch die resultierende Veränderung des projizierten Musters kann der Abstand zum Sensor ermittelt werden.

Durch die quadratisch zur Distanz sinkende Genauigkeit ist die Triangulation nicht für weit entfernte Objekte geeignet. Auch stellen starke Lichtquellen wie Sonnenstrahlung oder Arbeitsscheinwerfer sowie spiegelnde Flächen wie Sichtbeton oder Fenster eine enorme Fehlerquelle dar. Beim Einsatz mehrerer Triangulationssysteme stören diese sich auch gegenseitig. Die Technik der Triangulation ist damit insgesamt weniger für den Einsatz bei mobilen Robotern geeignet. Jedoch könnte sie ergänzend für den Nahbereich eingesetzt werden, da dort andere wellenbasierte Techniken wie LIDAR oder RADAR nicht mehr eingesetzt werden können (Konolige & Nüchter, 2016).

Bei der *Laufzeitmessung* (TOF) wird aus der Dauer eines ausgesendeten und wieder aufgenommenen Lichtsignals der zurückgelegte Weg ermittelt. Da Licht für eine Distanz von 30 Metern nur 10 Nanosekunden benötigt, sind auch weit entfernte Objekte präzise messbar. Es sind jedoch sehr genaue Zeitmessgeräte nötig und im Nahbereich weniger Zentimeter versagt diese Methode völlig, da die Messdauer zu kurz ist.

Die direkte Messung der Signallaufzeit wird durch ein Hochgeschwindigkeitschronometer realisiert. Da Laserlicht auch auf größere Distanz eine geringe Streuung hat wird dieses meist für direkte TOF verwendet. Ein solches System wird LIDAR (Light detection and ranging) oder LADAR (Laser RADAR) genannt. Der Abstand *d* eines gemessenen Punktes errechnet sich vereinfacht aus

$$2d = c \cdot t \tag{3.5}$$

wobei *c* die Lichtgeschwindigkeit und *t* die Signallaufzeit für Hin- und Rückweg ist. Da *c* abhängig vom durchquerten Medium ist (Luftfeuchtigkeit, Temperatur etc.) muss hier eine Korrektur anhand der aktuellen Umgebung durchgeführt werden. Die relevantere Fehlergröße ist hier aber die Zeitmessung. Diese hängt nicht nur von der Präzision des Zeitmessgerätes ab, sondern vor allem von der Erkennung des Signals selbst. Um ein Signal zu detektieren, versucht man üblicherweise das Maximum eines Signals zu finden. Durch Streuung und Reflexion wird das Maximum aber zeitlich gespreizt und wird schwerer als solches zu erkennen. Daher wird meist automatisiert ein Mittelwert aus mehreren kurz hintereinander ausgeführten Messungen ermittelt.

Für praktische Anwendungen bei mobilen Robotern ist ein einzelner Messpunkt nicht ausreichend. Daher werden meist Linienlaser zur Abtastung genutzt und über die abzutastende Oberfläche bewegt. Das Ergebnis ist dann ein Vektor mit Abständen der Scanebene zur Oberfläche.

Bei der praktisch immer gegebenen Durchführung mehrerer Messungen muss ab bestimmten Distanzen beachtet werden, dass ein Signal der vorherigen Messungen durch lange Laufzeit zu einem weit entfernten Objekt eventuell fälschlich der nächsten Messung zugeordnet wird. Konkret wird beispielsweise bei einer Abtastrate von 100 kHz, also 100.000 Messungen pro Sekunde, ab einer Objektdistanz von 1500 Metern ein Signal der nächsten Messung zugeordnet. Ist ein Objekt mehr als 3000 Meter entfernt, würde ein Signal der übernächsten Messung zugeordnet werden. Da die Abtastraten im geplanten praktischen Einsatz jedoch deutlich darunter liegen und auch die gemessenen Distanzen vergleichsweise klein sind, ist dieses Problem hier nicht relevant.

Bei der indirekten Laufzeitmessung werden bestimmte physikalische Eigenschaften des verwendeten Lichtstrahls ausgenutzt. Hauptsächlich kommen dabei Amplitudenmodulation oder Frequenzmodulation zum Einsatz. Beide Methoden nutzen dabei die messbare Veränderung zwischen gesendetem und empfangenen Signal zur Laufzeitmessung. Für die Modulation der Amplitude wird die Intensität s des Lichtsignals zeitlich verändert. Für eine Modulationsfrequenz f ergibt sich damit die von der Zeit abhängige Lichtintensität $s(t)$ zu

$$s(t) = \sin(2\pi f t) \tag{3.6}$$

Das Signal kommt unter einer Phasenverschiebung φ wieder am Sensor an. Die Intensität des zurückgeworfenen Signals $r(t)$ und daraus der Objektabstand d ergibt sich damit aus

$$r(t) = R\sin(2\pi f t - \varphi) = R\sin\left[2\pi f\left(t - \frac{2d}{c}\right)\right] \tag{3.7}$$

sowie

$$d = \frac{c\varphi}{4\pi f} \tag{3.8}$$

mit Lichtgeschwindigkeit c, Objektabstand d und Amplitude R des zurückgeworfenen Signals. Auch hier besteht, wie bei der direkten Messung der TOF, die Gefahr der Uneindeutigkeit des empfangen Signals. Das Intervall, in dem Uneindeutigkeiten auftreten können, hängt von der Modulationsfrequenz f ab und ergibt sich zu $c/2f$. Für beispielsweise $f = 10\,\mathrm{MHz}$ beträgt dieses Intervall $\Delta d \approx 15\,\mathrm{m}$. Da bei Amplitudenmodulation die effektive Reichweite gering ist und nur wenige Meter beträgt, sind diese Uneindeutigkeiten eher bei Grenzfällen zu beachten. Vorteil dieser Methode ist die extrem günstige Herstellbarkeit und Robustheit gegenüber Störquellen durch Sonnenlicht.

Demgegenüber steht die Modulation der Frequenz, was in einer *frequency modulation continuous wave* (kurz FMCW, kontinuierliche frequenzmodulierte Welle) resultiert. Dabei wird die Frequenz innerhalb eines Frequenzbereiches Δf während einer Zeitdauer t_m verändert. Aus der Signaldifferenz f_i zwischen gesendetem und empfangenem Signal ergibt sich die Distanz zum Objekt aus

$$d = f_i c t_m / 2\Delta f \tag{3.9}$$

Da Frequenzdifferenzen technisch sehr präzise messbar sind, kann mit dieser Technik eine hohe Genauigkeit der Tiefe erzielt werden. Laser-Licht lässt sich allerdings nur mit hohem Aufwand in seiner Frequenz modulieren, daher wird diese Technik nur in Geräten mit einzelnen Strahlen verwendet (Konolige & Nüchter, 2016; Shan & Toth, 2017).

3.2.3. LIDAR – LIGHT DETECTION AND RANGING

Die Methode des *Light Detection and Ranging* oder kurz LIDAR nutzt die im Kapitel 3.2.2 beschrieben Technik der TOF (Signallaufzeit) aus. Als Signal wird dabei meist ein Laser verwendet, weshalb auch manchmal LADAR (*Laser Detection and Ranging*) als Bezeichnung genutzt wird. Die benötigte Technik zur Erzeugung präzisen Laserlichtes sowie nötige Messgeräte zur Erfassung der Messsignale erfahren derzeit eine intensive Weiterentwicklung, da nicht nur im Bereich der Robotik sondern auch für autonome Fahrzeuge, Verkehrsüberwachung und andere Anwendungen Bedarf für präzise Umgebungsscans besteht (Woodside Capital Partners & Yole Développement, 2018). Dementsprechend scheint diese Technik aktuell und zukünftig am erfolgversprechendsten zu sein, eine ausreichend präzise und schnelle Umgebungsmessung zur Lokalisierung und Navigation eines mobilen Roboters bieten zu können.

Um eine Umgebung ausreichend abtasten zu können, muss eine Vielzahl Messungen durchgeführt werden. Wird dabei nur ein einzelnes Lichtsignal für Punktmessungen genutzt, muss dieses über die Oberflächen der Umgebung weiterbewegt werden. Eine Möglichkeit bietet die Nutzung eines oder mehrerer beweglicher Spiegel, die mechanisch gedreht werden und das Lichtsignal in die gewünschte Richtung reflektieren. Diese Methode birgt allerdings mehrere gravierende Nachteile. Durch die Verwendung mechanischer Bauteile besteht die Gefahr der Abnutzung der Mechanik und daraus resultierende Messungenauigkeiten mit steigendem Alter. Weiterhin sind Beschädigungen durch eindringenden Staub, Sand oder andere Kleinstteile wahrscheinlich. Da während der Errichtung eines Bauwerkes regelmäßig mit hohem Staubeintrag zu rechnen ist, müssten für eine Anwendung auf einer Baustelle besondere Schutzmaßnahmen für die Spiegelmechanik getroffen werden. Zusätzlich ist die Einbauhöhe der Spiegel mit mehreren Dezimetern entsprechend hoch und beschränkt damit möglicherweise die Nutzhöhe des mobilen Roboters (Shan & Toth, 2017; Voigt & Kremsreiter, 2020).

Die Verwendung einer elektromechanisch gesteuerten Linse anstatt von Spiegeln zur Bewegung des Lichtsignals kann die zuvor genannten Nachteile größtenteils ausgleichen. Daher kommen im Bereich der mobilen Roboter und autonomen Fahrzeuge immer häufiger sogenannte *MEMS-Mirrors* (Micro-Electro-Mechanical-Systems Mirrors) zum Einsatz. Durch die rein elektromechanische Bewegung der Spiegel ist das System deutlich weniger anfällig gegenüber Staubeintrag und deutlich kleiner. Dadurch kann die Einbaugröße des gesamten LIDAR-Systems wesentlich kleiner ausfallen. Nachteile ergeben sich durch die vergleichsweise hohe Anfälligkeit für Temperaturschwankungen, die zumindest beim Einsatz auf Baustellen immer signifikant vorhanden sind. Eine regelmäßige Kalibrierung des Systems kann dadurch notwendig werden. Auch Vibrationen und Erschütterungen haben einen starken Einfluss auf die Messgenauigkeit, was bei einem mobilen Einsatz fast immer relevant ist. Daher ist möglicherweise eine zusätzliche Schwingungs- und Stoßdämpfung des LIDAR-Systems nötig.

Um ein sofortiges Abbild der Umgebung zu erhalten, sind *non-scanning LIDAR-Flashs* verfügbar. Diese senden ähnlich einem Kamerablitz ein stark gestreutes Lichtsignal in alle Richtungen gleichzeitig aus. Das gesamte Sichtfeld wird somit in einer einzelnen Messung vollständig ausgeleuchtet und nicht schrittweise gescannt. Um ausreichend stark von der Umgebung reflektierte Signale zu erhalten, muss durch die gleichzeitig notwendige Streuung besonders energieintensives Laserlicht genutzt werden. Daher müssen hier Vorkehrungen getroffen werden, damit es nicht zu Augenschäden anwesender Personen kommt. Die sofortige Aufnahme auch während stärkerer Lageveränderungen des mobilen Roboters machen diese Methode allerdings dennoch vielversprechend (Shan & Toth, 2017; Voigt & Kremsreiter, 2020).

3.2.4. INDIREKTE ENTFERNUNGSMESSUNG

Indirekte Methoden der Entfernungsmessungen nutzen zusätzliche Informationen, um Entfernungen zu ermitteln. Dazu zählen das Ermitteln des zurückgelegten Weges aus Messungen der ausgeführten Radumdrehungen, Beschleunigungs- und Neigungssensoren. Da anhand dieser Informationen nur die relative Position und Entfernung zur Ausgangsposition ermittelt werden kann, muss zusätzlich bereits eine Umgebungskarte sowie die Ausgangsposition innerhalb der Karte bekannt sein. Da eine neue, indirekt ermittelte Position immer von der vorherigen abhängt, vervielfältigen sich Messfehler immer weiter. Diese treten zum Beispiel signifikant durch Schlupf der Räder auf. Insbesondere in engen Kurven oder bei losem Untergrund kommt es zu großen Ungenauigkeiten. Daher sind indirekte Messungen eher in Kombination oder ergänzend zur Validierung anderer Messsysteme geeignet. Die Lokalisierung anhand der Signalstärke von externen Signalquellen wie WLAN, Bluetooth oder Mobilfunk ist ebenfalls möglich und wird in Abschnitt 3.2.5 betrachtet. So kombinieren beispielsweise Park et al. Beschleunigungs- und Neigungssensoren sowie Magnetkompass mit einem eingemessenen Funknetzwerk aus UWB-Sendern.

3.2.5. FUNKNETZWERKE

Eine spezielle Methode der indirekten Entfernungsmessung ist das Nutzen von Funknetzwerken. Dabei wird anhand der am mobilen Roboter eingehenden Signalstärken mehrerer Signalquellen die Position zwischen den Sendern ermittelt. Die Position der Sender muss dabei a priori bekannt sein. Dafür kommen beispielsweise WLAN, UWB (Ultra-

Breitband oder englisch ultra-wideband), Bluetooth, RFID (Radio Frequency Identification), Ultraschall oder Infrarot in Frage.

Besonders UWB ist durch den geringen Energieverbrauch, der Fähigkeit zur Durchdringung von Wänden und der hohen Genauigkeit besonders für diesen Zweck geeignet. Es dürfen dabei - abhängig von nationalen Gesetzgebungen - deutlich höhere Signalstärken als beispielweise bei Bluetooth oder WLAN genutzt werden, was die Reichweite dieser Funktechnik gegenüber anderen erhöht. Die effektive maximale Reichweite eines Senders liegt dabei bei circa 100 Metern ohne Hindernisse. Vor allem in Innenräumen kann diese durch Reflexion und Abschirmung durch Wände, Wasserrohre und andere Hindernisse bis auf wenige Meter sinken (Park et al., 2016, S. 31–33; Siciliano & Khatib, 2016, S. 661).

Besonders problematisch ist bei Funknetzwerken auf Baustellen die fast nie vorhandene Funk-Infrastruktur. Selbst einfacher Mobilfunkempfang ist nicht immer ausreichend verfügbar. UWB bietet allerdings eine hohe lokale Datenübertragungsrate. Daher könnte eine zusätzliche UWB-Funk-Infrastruktur bei größeren Bauprojekten als allgemeiner Kommunikationskanal fungieren. Auch mobile Geräte wie Tablets oder Smartphones, aber auch ein mobiler Roboter können mit dieser Technik ausgestattet werden. Somit besteht die Möglichkeit, jeglichen lokalen Datenverkehr über Funknetzwerke im UWB-Bereich abzuwickeln. Eine große Schwierigkeit wird allerdings darin bestehen, die Position der Sender ausreichend genau zu ermitteln und diese während des Baufortschritts dauerhaft mit Strom zu versorgen und vor Beschädigung zu schützen. Auch die Befestigung der Sender muss so beschaffen sein, dass sie einerseits sicher an ihrer Position gehalten werden, gleichzeitig aber flexibel bewegt werden können, um Arbeiten in unmittelbarer Nähe nicht zu stören.

Da vor allem eine schnelle und kostengünstige Montage, Einmessung und Versorgung eines Funknetzwerkes in naher Zukunft nicht wahrscheinlich ist und mit anderen Methoden der Lokalisierung bereits Erfolge erzielt werden konnten (Deng et al., 2017; Ha et al., 2018; Nahangi et al., 2018; Neges et al., 2017; Tashakkori et al., 2015), wird dieser Ansatz hier nicht weiter verfolgt.

3.3. NAVIGATIONSMODELLE

Durch die in Kapitel 3.2 beschriebenen Techniken zur Entfernungsmessung wird der aktuelle, relativ zur unmittelbaren Umgebung des Roboters befindliche Standort ermittelt. Um mithilfe dieser Information einen globalen Standort zu ermitteln sowie Wege planen zu können, wird ein sogenanntes Navigationsmodell benötigt. Ein Navigationsmodell ist eine explizite Repräsentation geometrischer, topologischer und semantischer Informationen der physikalischen Umgebung (Taneja et al., 2016, S. 24). Die in dieser Diplomarbeit betrachteten Bezugsräume lassen sich dabei als eine solche physikalische Umgebung auffassen und auf verschiedene Art und Weise repräsentieren: basierend auf der Mittelachse der Bezugsräume, den Raumgrenzen oder der Nutzung von Sichtlinien im Innenraum (Karimi & Iordanova, 2021, S. 15).

3.3.1. MITTELACHSENBASIERT

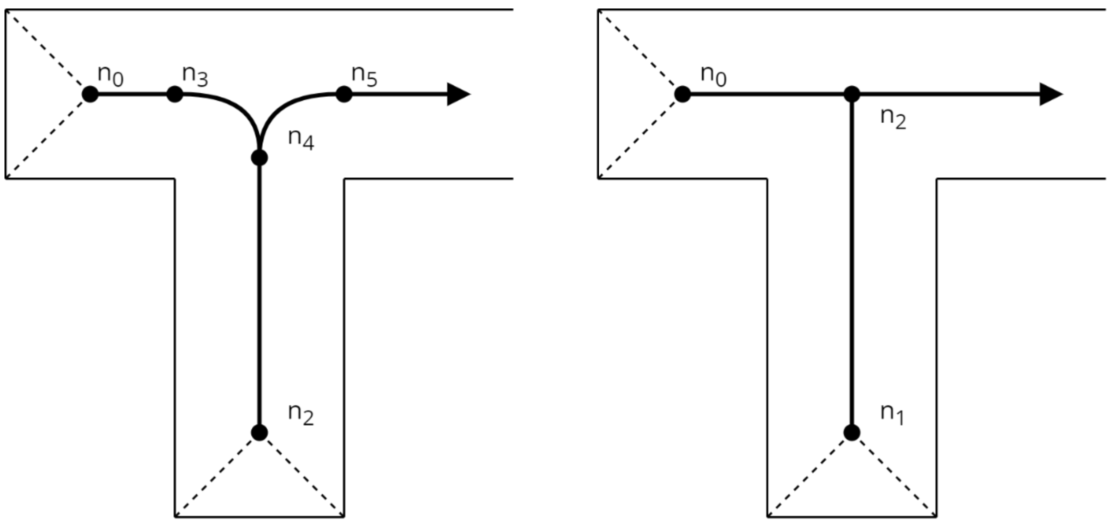
Bezugsräume sind in der Realität zweidimensionale, ebene Vielecke (planare Polygone), deren Kanten sich nur in den Eckpunkten schneiden. Falls dagegen beispielweise Wände oder Säulen gekrümmte Geometrien aufweisen, sind die Kanten nicht mehr gerade, sondern werden als Kurven beschrieben. In jedem Bezugsraum hat ein Fahrzeug oder Fußgänger viele Freiheitsgrade, diesen zu durchqueren. Durch diese Vielzahl an Möglichkeiten ist eine Wegberechnung nur mit hohem Speicher- und Rechenaufwand zu leisten und nicht praktikabel (Taneja et al., 2016). Um den Suchraum der möglichen Lösungen für Navigationspfade und Lokalisierung auf wenige Lösungsmöglichkeiten einzuschränken, müssen die Bewegungs-Freiheitsgrade entsprechend verringert werden. Dies kann durch eine geometrische Näherung der Bezugsräume erreicht werden, indem man die zweidimensionale Raumgeometrie auf eine eindimensionale Geometrie abbildet. Da diese Abbildung auf die Mittelachse einem Skelett ähnelt, wird der Vorgang auch *topologische Skelettierung* und die Abbildungsvorschrift Skelettfunktion genannt. Eine 1967 von H. Blum veröffentlichte Publikation charakterisiert eine solche Skelettlinie als „zwischen einer reinen Topologie und reiner geometrischer Darstellung liegend“. Sie enthält im Gegensatz zu reiner Topologie noch geometrische Informationen über die Form des Objekts (Blum, 1967, S. 368).

Das aus einer Skelettierung entstehende Navigationsmodell ist vergleichbar mit einem Straßennetz, wie es in der Fahrzeugnavigation auch verwendet wird und in dem sich nur auf den Skelettlinien und ausschließlich entlang einer Dimension bewegt werden kann. Wege werden durch Kanten und Entscheidungsmöglichkeiten beziehungsweise Wegkreuzungen durch Knoten der Skelettlinie repräsentiert (Karimi & Iordanova, 2021).

Eine große Schwierigkeit besteht dabei in der Erzeugung der Mittelachsen beziehungsweise Skelettlinie von Bezugsräumen. Aus dem von Blum (1967) vorgeschlagenen Algorithmus der *Medial Axis Function* (MAF) wurden diverse praktisch anwendbare Abbildungsvorschriften entwickelt und verbessert. Dazu eingesetzte Algorithmen basieren meist auf der 1908 von Georges Voronoi[8] veröffentlichten Methode der *Voronoi Tesselation*, welche für eine definierte Menge Quellpunkte in einer Ebene den minimalen Abstand jeder dieser Punkte zueinander ermittelt. Die daraus entstandene Zerlegung der Ebene besteht aus

---

[8] Der ukrainisch-russisch-stämmige Mathematiker publizierte viele seiner Werke auf Französisch, weshalb meist die französische Transkription des Namens verwendet wird. Seine von ihm selbst genutzte, transkribierte Schreibweise lautet Georgy Theodosiyovych Voronoï.



MAT-Algorithmus S-MAT-Algorithmus

Abbildung 6 - Vergleich des MAT- und S-MAT-Algorithmus an einer Kreuzung

sogenannten *Voronoi-Zellen* und deren jeweiligen Quellpunkten im Mittelpunkt. Das Ergebnis einer solchen Zerlegung wird auch *Voronoi Diagramm* genannt (Syta & van de Weygaert, 2009; Voronoi, 1908).

Um die Berechnung der Mittelachse zu beschleunigen, stellte D.-T. Lee (1982) den MAT-Algorithmus (Medial Axis Transformation) vor, der das Polygon nach dem *Divide-and-Conquer-Prinzip* in immer kleinere Teilpolygone trennt. Anschließend wird ausgehend von der Winkelhalbierenden zwischen den Ecken der Teilpolygone das Voronoi Diagramm für die Teilpolygone ermittelt. Im letzten Schritt werden die einzelnen Voronoi Diagramme wieder zusammengeführt. Die Skelettlinie besteht dann aus den Kanten des kombinierten Voronoi Diagramms abzüglich der Winkelhalbierenden konkaver Ecken. Konkav bedeutet hierbei, dass der Innenwinkel zwischen zwei Polygonkanten größer als 180 Grad ist (D.-T. Lee, 1982). Ein bedeutender Nachteil ist allerdings, dass hierbei oft komplexe Kombinationen aus Geraden und Parabelkurven als Skelettlinie entstehen, die nur umständlich nutzbar sind. Zusätzlich ist die Zusammenführung der Teildiagramme nichttrivial zu implementieren (J. Lee, 2004; Taneja et al., 2016, S. 28).

Eine Abwandlung des MAT-Algorithmus ist der *Straight-MAT-Algorithmus*, kurz S-MAT. Dieser erzeugt statt Kurvensegmenten ausschließlich gerade Abschnitte, um vor allem Wegkreuzungen präziser darstellen zu können, wie in Abbildung 6 dargestellt. An H-förmigen Kreuzungen wie in Abbildung 7 kommt es allerdings zum Versagen des S-MAT-Algorithmus, da dieser nur von konvexen Ecken ausgehend Winkelhalbierende bildet und daher an den Knoten n_1 beziehungsweise n_2 stoppt. Um die Nachteile des S-MAT-Algorithmus auszugleichen und die Vorteile zu erhalten, stellten Taneja et al. (2016) einen modifizierten MAT-Algorithmus vor (M-MAT). Dieser bietet für jede Art Wegkreuzung eine Lösung,

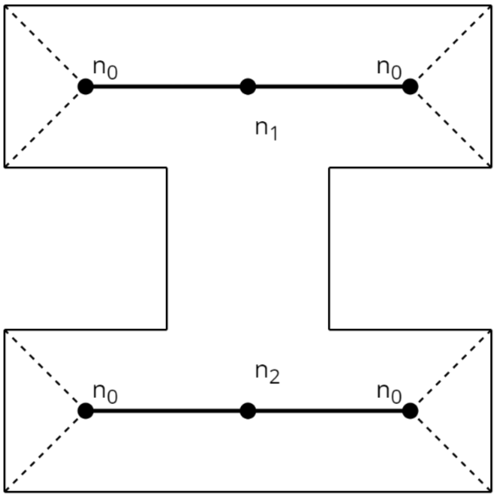


Abbildung 7 - Versagen des S-MAT-Algorithmus an H-förmiger Kreuzung

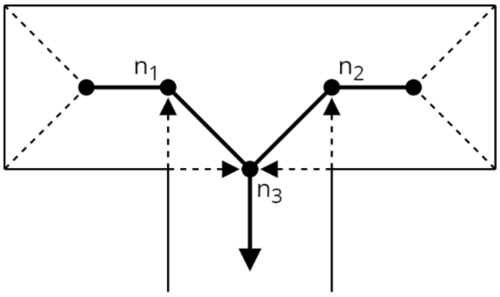


Abbildung 8 - Lösung des M-MAT-Algorithmus an konkaven Ecken

für einige Kreuzungsarten aber keinen optimalen, geraden Pfad. In Abbildung 8 ist zu erkennen, dass zwar ein Pfad nach unten gefunden wird, was jedoch gleichzeitig den Weg von Knoten n_1 zu n_2 verlängert (J. Lee, 2004, S. 244; Taneja et al., 2016, S. 28).

Die Genauigkeit der Nutzung einer Mittelachse oder Skelettlinie als Navigationspfad ist daher stark abhängig von der Geometrie und Verteilung von Raumecken sowie dem Verhältnis der Länge der Raumbegrenzung zur Raumfläche. Je allgemeingültiger ein Algorithmus eine Mittelachse erzeugen kann, desto eher kommt es zu einer verzerrten Repräsentation der Raumgeometrie. Großflächige Räume müssen zuvor in kleinere Teilräume getrennt werden, um eine möglichst genaue Abdeckung der Raumfläche zu erhalten.

3.3.2. METRIKBASIERT

Anstatt sich an der Skelettlinie eines Raumes zu orientieren, ist auch eine Nutzung der Begrenzungen der Bezugsräume möglich. Aus jedem physischen Objekt eines IFC-Modells lässt sich eine Repräsentation der Begrenzungen dessen Bezugsraumes generieren. Dies sind beispielsweise Wände, Stützen oder Türen. Jedem Bezugsraum wird außerdem zugeordnet, ob er durchdringbar oder nicht durchdringbar ist. Zusätzlich muss in einem solchen Navigationsmodell ein globales Koordinatensystem vorliegen, um eine Wegplanung von einem Bezugsraum zu einem anderen durchführen zu können. Kreuzt ein Weg einen nicht durchdringbaren Bezugsraum, wird entlang der Begrenzung navigiert, bis ein Durchgang möglich ist.

Rasterung
Um auch innerhalb von Bezugsräumen Wege zuzulassen, können künstliche Raumgrenzen eingeführt werden. Dies lässt sich durch Felder eines Gitter oder Netzes (Rasterung) realisieren, welche die gesamte Fläche des Bauwerkes abdecken. Die Erzeugung der Felder kann dabei entweder durch ein Gitter aus regelmäßigen Feldern oder durch ein Netz aus unregelmäßigen Feldern geschehen. Die Nutzung eines gleichmäßigen Gitters hat den Vorteil, dass das Generieren und Speichern mit geringem Aufwand möglich ist. Um allerdings eine ausreichend genaue Näherung der realen Raumgeometrie zu erhalten, darf die Feldgröße nicht zu groß gewählt werden. Eine generell gültige Feldgröße kann es dabei nicht geben. Sie ist abhängig vom Detailgrad der Bezugsräume und wie weit entfernt voneinander Durchgänge, Ecken, Hindernisse etc. sind; wie groß der zu navigierende Roboter ist und wie leistungsstark das Gesamtsystem zur Berechnung des optimalen Weges ist. Ein zu feines Netz resultiert in eine nicht mehr praktisch nutzbare Rechendauer zur Wegermittlung, während ein zu grobes Netz eine ungenaue Positionierung und Wegplanung sowie eine ungenaue Raumabdeckung bedingt. In den Veröffentlichungen von Taneja et al. (2016) und Kerbitz (2019) wird als Kompromiss zwischen Genauigkeit und erwarteter Rechenzeit beispielsweise eine Rasterung durch Quadrate mit einer Seitenlänge von 50 Zentimeter genutzt (Kerbitz, 2019; Taneja et al., 2016; Yuan & Schneider, 2010).

Denkbar ist auch eine a-priori-Optimierung des Rasters in einem leistungsstarken Rechenzentrum. Das optimierte Raster könnte anschließend an den Roboter übergeben werden, um mit dessen geringerer, lokal verfügbarer Rechenleistung ausreichend genaue Feldgrößen nutzen zu können. Dazu muss automatisiert mit jeder Aktualisierung des BIM-Modells eine Optimierung des Rasters durchgeführt werden. Das resultierende

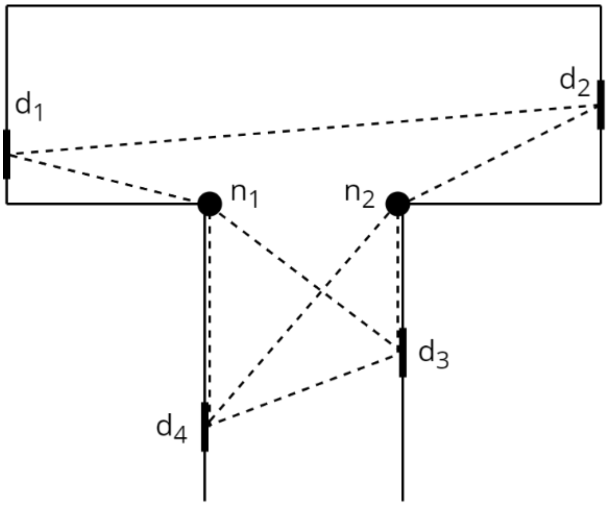


Abbildung 9 - Sichtlinienbasiertes Navigationsmodell

Navigationsmodell könnte dann beispielsweise immer zu Beginn des Arbeitstages oder immer dann, wenn der Roboter zur Ladestation fahren muss, mit diesem synchronisiert werden.

3.3.3. SICHTLINIENBASIERT

Ein Navigationsmodell basierend auf Sichtlinien ähnelt dem eines metrikbasierten, ist allerdings nicht auf Raumgrenzen beschränkt. Die Knoten in einem solchen Navigationsmodell stellen sogenannte Haltepunkte dar, an denen entweder ein Raum durch zum Beispiel eine Tür betreten wird, oder eine Sichtlinie aufhört. Eine Sichtline entsteht, wenn eine direkte Verbindung zwischen zwei Haltepunkten nicht möglich ist, weil eine Wand oder ein anderes Hindernis dazwischen liegt, wie in Abbildung 9 dargestellt. Die Haltepunkte n_1 und n_2 sind durch die Begrenzung der Sichtlinien von den Türen d_1 zu d_3 und d_4 beziehungsweise von d_2 zu d_3 und d_4 entstanden. Eine Implementierung dieser Methode stellten Liu und Zlatanova (2011) als „Tür-zu-Tür-Navigation“ vor, welche zwar durch die komplizierte Interpretierbarkeit zur Innenraum-Navigation von Menschen ungeeignet ist (W. Y. Lin et al., 2017), für Roboter aber nützlich sein kann.

Eine Kombination aus einem sichtlinienbasierten Navigationsmodell bis zum Haltepunkt, der zum Raum des Einsatzortes führt mit einem rasterbasierten Navigationsmodell nahe des Einsatzortes könnte ebenfalls ein Ansatz sein, der die rasterbasierte Genauigkeit mit der schnelleren Rechenzeit und kürzeren Wegen von sichtlinienbasierten Navigationsmodellen vereint. In Abbildung 10 ist beispielsweise eine Navigation von Tür d_1 zu Tür d_3 via Sichtlinie vorgesehen, während im Zielraum ab Tür d_3 entlang eines Rasters bis zum Zielpunkt n_G navigiert wird.

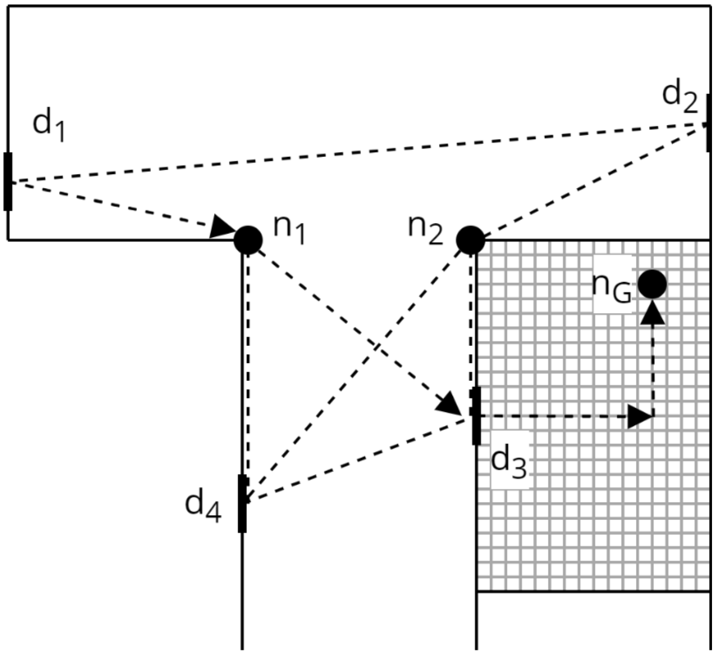


Abbildung 10 - Kombination aus Rasterung und Sichtlinien

3.4. UMGEBUNGSREPRÄSENTATION IN ROS

Im *Robot Operating System* besteht die Möglichkeit, die Umgebung durch eine oder mehrere zweidimensionale Karten zu repräsentieren. Diese können von einfachen, reinen Geometriedarstellungen bis hin zu umfangreichen, mehrschichtigen Datensammlungen mit beispielsweise Höhenangaben oder Geschwindigkeitsvorgaben sein. Eine Schicht einer Karte wird *Layer* genannt und kann nach Bedarf mit anderen Schichten kombiniert werden (Macenski, Martín et al., 2020).

3.4.1. FLÄCHIGE GEOMETRIEREPRÄSENTATION

Die einfachste Kartenschicht in ROS stellt eine Geometrierepräsentation durch einen Grundriss oder Horizontalschnitt der Umgebung dar. Dazu wird eine zweidimensionale Bilddatei in Graustufen in Kombination mit einer strukturierte Textdatei in der Auszeichnungssprache YAML (Akronym für *YAML Ain't Markup Language*, zuvor *Yet Another Markup Language*) genutzt.

Die Bilddatei wird ROS-intern im Bildformat PGM[9] als Pixelgrafik verarbeitet. Es handelt sich dabei um eine zweidimensionale Zahlen-Matrix mit beliebigen, positiv ganzzahligen Werten für jedes Pixel, welche die Intensität der Grauwerte widerspiegeln. Meist wird dafür der Zahlenbereich von 0 bis 255 gewählt, da dies dem häufig genutzten 8-Bit-Standard für Graustufenbilder entspricht. Ist eine feinere Abstufung nötig, wird auf 16-Bit-Graustufen im Bereich von 0 bis 65535 zurückgegriffen. Die Intensität repräsentiert die Wahrscheinlichkeit, mit der eine Fläche auf der Karte „belegt" und damit unbegehbar ist. Ein Grauwert von 0 entspricht einem weißen Pixel und damit einer freien Fläche in der Realität, während ein hoher Grauwert auf eine belegte (schwarze) Fläche hindeutet.

Die YAML-Datei enthält Metainformationen zur Karte, um die Bilddatei eindeutig interpretieren zu können. Die Auflösung der Karte wird dabei als Längeneinheit pro Pixel angegeben und hängt stark vom Anwendungsfall und der Umgebungsgröße ab. Wählt man beispielsweise für eine Grundfläche von 30 Meter Länge und 80 Meter Breite eine Auflösung von 5 Zentimetern je Pixel, muss das zugehörige Bild 600 mal 1600 Pixel groß sein. Die zu speichernden 960.000 Pixel benötigen je nach Einstellung von Codierung der Datei ca. 3,7 Megabyte Speicherplatz, was auch auf leistungsschwacher Hardware kein Problem darstellt. Eine Auflösung von 1 Zentimeter je Pixel würde jedoch bereits 24 Millionen Pixel benötigen, was eine Dateigröße von circa 89 Megabyte bedingt. Das kann auf portablen Systemen bereits zu Leistungsproblemen führen, da Rechenleistung und Arbeitsspeicher zum Verarbeiten der Bilddatei nicht immer auf derartige Datenmengen ausgelegt sind.

---

[9] PGM steht für Portable Grey Map und ist im Open-Source-Softwarepaket *Netbpm* definiert. Weitere Informationen sind unter
netpbm.sourceforge.net/ und netpbm.sourceforge.net/doc/pgm.html zu finden (Website jeweils zuletzt besucht am 03.08.2021).

Eine individuelle Anpassung der Kartenauflösung abhängig von verfügbarer Hardware, Einsatzzweck und Beschaffenheit der Arbeitsumgebung ist daher unumgänglich.

Weiterhin muss als Metainformation die Lage des Koordinatenursprungs im Bild definiert werden. Außerdem muss der Schwellwert der Graustufe, ab der ein Feld als „belegt“ oder als „frei“ gilt, angegeben sein (Quigley et al., 2015, S. 136–137). Während des Robotereinsatzes kann eine solche Umgebungskarte auch durch vom Roboter gewonnene Informationen ergänzt werden. Diese Methodik wird in Abschnitt 3.6 erläutert.

3.4.2. COSTMAPS UND LAYER

Die in Abschnitt 3.4.1 vorgestellte Repräsentation der Umgebungsgeometrie stellt allgemein eine *Costmap* dar, da sie den notwendigen Aufwand beziehungsweise die „Kosten“ widerspiegelt, eine bestimmte Aktion auszuführen, wie beispielsweise ein bestimmtes Feld zu Befahren. Die Definition von „Kosten“ einer Aktion ist dabei abhängig von deren Ziel. Für eine Navigation durch ein Gebäude kommen zum Beispiel die Fahrtdauer, Energieverbrauch, mögliche Verlangsamung durch Hindernisse oder Geschwindigkeitsbeschränkungen als zu beachtende Kosten in Frage. Um entsprechende Kostenfunktion errechnen zu können sind mehr Informationen als die reine, ebene Geometrie der Umgebung nötig. Weiterhin weist die Geometrie aus Bilddateien beispielsweise durch Modellfehler oder Komprimierung eine gewisse Unschärfe auf und auch Umgebungssensoren liefern nur unscharfe Daten. Daher ist eine einzige Kartenschicht mit der bloßen Information „frei“ oder „belegt“ im praktischen Einsatz meist unzureichend. Eine Karte in ROS besteht daher fast immer aus mehreren Kartenschichten (Layer) mit verschiedenen Costmaps.

Beispiele für weitere Costmaps sind (Höchst-) Geschwindigkeitskarten, Gradientenkarten (Angaben zu Steigung) oder Markierungen für besondere Orte wie zum Beispiel Ladestationen, Lagerplätze oder bevorzugte Fahrstrecken für Roboter. Auch eine Erweiterung für dreidimensionale Navigation ist möglich. Dazu kommen neben dreidimensionalen Costmaps unter anderem auch *Mesh Maps* zur dreidimensionalen Oberflächenmodellierung zum Einsatz. Diese benötigen allerdings deutlich höhere Speicherkapazität und Rechenleistung sowie andere Navigationsalgorithmen (Macenski, Delsey & White, 2020; Macenski, Martín et al., 2020).

3.5. UMGEBUNGSKARTEN AUS BIM-MODELLEN

Die in den Kapiteln 3.3 und 3.4. beschriebenen Navigationsmodelle und Umgebungsrepräsentationen werden im Bereich des Baugewerbes hauptsächlich aus folgenden Informationen gespeist (Siemiątkowska et al., 2013, S. 376–377):

- Koordinaten

- Voxel (*Vo*lume Pi*xel*, entspricht einem dreidimensionalen Bildpixel)
- Topologische Zusammenhänge
- Semantische Informationen
- Höhendaten

In einem mithilfe von BIM geplanten Bauwerk im Hochbau befinden sich fast ausschließlich horizontal orientierte ebene Abschnitte, wie sie auch in Abschnitt 2.3.5 als *spatial structure* definiert sind. Diese lassen sich semantisch und topologisch gliedern. Daher sind lokale und globale Koordinaten in entsprechenden Koordinatensystemen in Kombination mit topologischen und semantischen Informationen aus dem BIM-Modell als Kartenmaterial sinnvoll.

Die Autoren Karimi und Iordanova stellten in einem umfangreichen Literature Review fest, dass bisherige Publikationen einen oder mehrere der folgenden Ansätze zur Erzeugung einer Karte aus einem BIM-Modell nutzen (Karimi & Iordanova, 2021, S. 13–16):

- Rein geometrisch: Erzeugung eines „Grundrisses" durch einen horizontalen Schnitt durch das rein geometrische Gebäudemodell beziehungsweise die geometrische Repräsentation des BIM-Modells
- Topologisch: Überführen der *spatial structures* eines BIM-Modells in eine topologische Repräsentation des Gebäudes, zum Beispiel als Graph

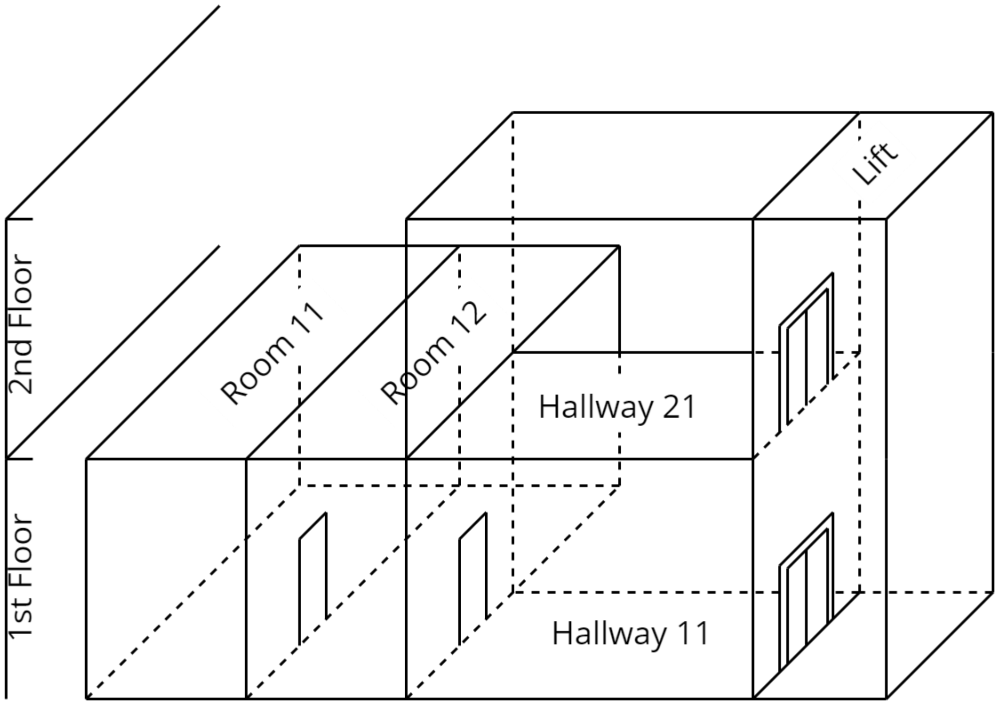


Abbildung 11 - Schematische Darstellung eines Beispielgebäudes

- Fotografisch: Erzeugung eines digitalen Abbildes des Gebäudeinneren als detailliertes Rendering, um Bilderkennung zur Orientierung nutzen zu können
- Ergänzung durch Semantik: Anreichern der Karte durch semantische Informationen wie Nutzungszweck der Räume oder Terminplanungen

In den folgenden Abschnitten werden diese Ansätze anhand bisheriger Veröffentlichungen erläutert. Dazu wird auf ein vereinfachtes Beispielhaus zurückgegriffen, welches in Abbildung 11 dargestellt ist.

3.5.1. ANSATZ 1: HORIZONTALER GEOMETRISCHER SCHNITT

Die Autoren Follini et al. (2021) nutzen einen dreiteiligen Algorithmus, um eine Karte zur Navigation und Lokalisierung des Roboters im Gebäude zu erhalten. Durch einen horizontalen Schnitt durch das gesamte Gebäude wird dessen grundlegender geometrische Aufbau repräsentiert. Die so generierte, globale Umgebungskarte wird durch eine Transformation des aktuellen Referenzsystems des Roboters in die globale Karte durch den sogenannten *AMCL-Algorithmus* (Adaptive Monte Carlo Localization Algorithmus) zur Lokalisation verwendet. Dies ermöglicht auch bei Vorhandensein kleinerer, unbekannter Objekte im Raum eine Erkennung der Raumgeometrie durch den Roboter (Follini et al., 2021, S. 2).

Zusätzlich wird die Umgebungskarte durch Informationen über temporäre Hindernisse im Gebäude angereichert, die aus dem BIM-Modell gewonnen werden. Dies sind beispielsweise temporär offene Bodenabdeckungen oder nicht befahrbare Kabelschächte. Eine solche Informationsanreicherung setzt entsprechend detaillierte BIM-Modelle voraus, führt aber im Rahmen dieser Diplomarbeit zu weit. Follini et al. (2021) nutzten dafür eigene Parameter, sogenannte *Custom Parameter*, da die von ihnen genutzte IFC Version 2x3 noch keine standardisierten Klassen dafür anbietet. Ab IFC 4 ist dafür die abstrakte Klasse *IfcSchedulingTime*[10] und deren Spezialisierungen vorgesehen (Follini et al., 2021, S. 4).

Der zuvor erwähnte dreiteilige Algorithmus zur Geometrierepräsentation ist hier von besonderer Bedeutung. Er besteht aus Analyse und Filterung der IFC-Datei, dem eigentlichen horizontalen Schnitt durch die Gebäudegeometrie und der anschließenden Kartenerzeugung und -darstellung.

Analyse der IFC Datei und Filterung relevanter Objekte
Im ersten Schritt wird die IFC-Datei semantisch analysiert und gefiltert. Konkret wird eine Unterscheidung zwischen physisch und nicht physisch vorhandener Objekte durch die in

---

[10] Siehe standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML/link/ifcschedulingtime.htm (Zuletzt abgerufen am 20.06.2021)

IFC definierte Klasse *IfcProductRepresentation*[11] vorgenommen. Diese gibt an, ob und welche physikalische Repräsentationen ein Objekt im Modell hat. Falls eine solche Repräsentation existiert, wird weiterhin geprüft, ob das Objekt im Bewegungsraum des Roboters liegt. Dazu wird die niedrigste z-Koordinate der Bounding Box des Objektes mit der höchsten z-Koordinate des Roboters verglichen. Somit können für den Roboter irrelevante Objekte wie beispielsweise Deckenplatten, hohe Überstände oder Unterzüge mit geringem Aufwand erkannt und herausgefiltert werden. Zusätzlich wird nach Objekten ohne Repräsentation auf Bodenhöhe gefiltert, da diese potentielle Löcher oder Öffnungen im Boden sein können. Diese fließen als unpassierbares Hindernis in die Navigation ein. Eine zusätzliche Filterung nach der *Zeit* wird ebenfalls vorgenommen. Ein Objekt wird ab dessen *Startdatum* errichtet oder erzeugt und ist bei dessen *Enddatum* fertig gestellt. Dabei gelten Objekte ohne Repräsentation zwischen deren Start- und Enddatum als Hindernis, während Objekte mit Repräsentationen dauerhaft ab deren Startdatum als Hindernis angesehen werden.

Querschnitt durch Gebäudegeometrie
Nur die nach der Filterung verbliebenen Objekte werden für den horizontalen Schnitt in Betracht gezogen. Der Schnitt wird auf Höhe der Umgebungssensoren des Roboters durchgeführt, was den Abgleich zwischen durch die Sensoren aufgenommenen Daten der Umgebung mit der aus dem BIM-Modell erstellten Karte verbessert. Die technische Umsetzung erfolgte durch das Softwarepaket *Open CASCADE Technology* (kurz OCCT) in Verbindung mit der Python-Schnittstelle *pythonOCC* (Open Cascade S.A.S, 2020; Paviot, 2021). Um das Ergebnis vorläufig zu prüfen, wurde außerdem eine manuelle Sichtprüfung im

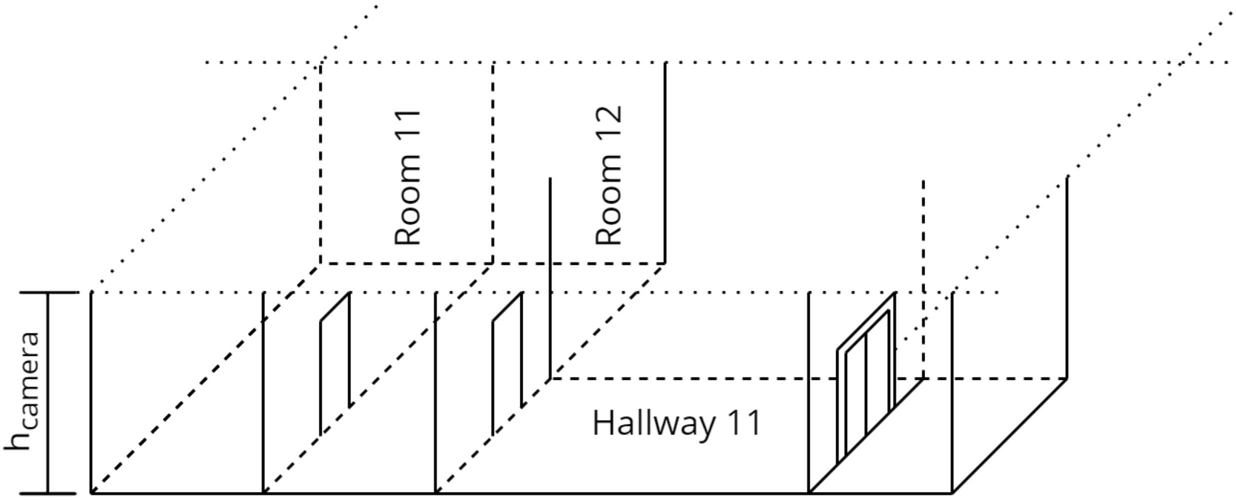


Abbildung 12 - Horizontaler Schnitt durch Gebäudegeometrie

---

[11] Siehe standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML/schema/ifcrepresentationresource/lexical/ifcproductrepresentation.htm (Zuletzt abgerufen am 20.06.2021)

integrierten Viewer in OCCT durchgeführt. In Abbildung 12 ist ein solcher Schnitt schematisch am Beispielgebäude dargestellt.

Erstellung der Kartendatei
Der zuvor erstellte horizontale Schnitt wird als PGM-Bilddatei (siehe Abschnitt 3.4) gespeichert und kann anschließend als Karte für den in ROS programmierten Roboter genutzt werden. Dabei wurden ausschließlich binäre schwarz-weiß-Bilder erzeugt. Das bedeutet, dass für den Roboter ein Feld entweder sicher als „belegt“ oder sicher als „frei“ gilt. Unsicherheiten durch Abweichungen bei der Bauausführung oder im BIM-Modell nicht enthaltene kleinere Objekte werden später während des praktischen Einsatzes durch die LIDAR-Sensoren des Roboters erkannt und mithilfe des zuvor erwähnten AMCL-Algorithmus mit der Kartendatei abgeglichen (Follini et al., 2021, S. 6).

Die zum Einsatzzweck nötigen Informationen werden „vor Arbeitsbeginn“ jeweils aus dem IFC Export abgerufen (Follini et al., 2021, S. 3). Wie die Übertragung im Detail stattfindet, wird nicht spezifiziert. Die erstellten Karten sind jeweils nur für begrenzte Gebiete innerhalb einer Etage nutzbar. Navigation zwischen verschiedenen Etagen ist durch die Beschränkung auf einen zweidimensionalen Schnitt nicht möglich.

3.5.2. ANSATZ 2: TOPOLOGISCHES MODELL

Die Nutzung eines rein geometrischen Modells führt bei großen, komplexen Gebäudegeometrien zu hohem Speicher- und Rechenbedarf. Außerdem müssen semantische Anweisungen erst in geometrische Ortsangaben übersetzt werden, was eine zusätzliche Kartierung notwendig macht: soll beispielsweise „zum Raum 11“ navigiert werden, muss eine Information vorliegen, an welchen Koordinaten sich ein Eingang zu „Raum 11“ befindet und durch welche anderen Räume man dorthin gelangt.

Ein alternativer Ansatz ist daher die Nutzung eines topologischen Modells des Gebäudes. Üblicherweise werden räumliche Strukturen als Graph repräsentiert. Räume werden dabei durch Knoten und Verbindungen zwischen diesen durch Kanten repräsentiert. Zusätzlich lassen sich die Verbindungen wichten, wodurch sich beispielsweise Abstände oder benötigte Fahrzeit zwischen zwei Räumen modellieren lassen. Um von Raum zu Raum navigieren zu können, kann der kürzeste oder schnellste Weg von Start- zu Zielknoten ermittelt werden, wofür leistungsfähige Algorithmen zur Verfügung stehen. Ein analoges Prinzip wird auch in der Navigation von Straßenfahrzeugen genutzt, wobei Ortschaften, wichtige Orte oder Kreuzungen als Knoten und die Straßen dazwischen als Kanten repräsentiert werden.

Hypergraph als topologisches Modell
Die Autoren Palacz et al. (2019) erzeugen dazu über mehrere Konvertierungsschritte einen Hypergraphen, ähnlich dem in Abbildung 13. Die Autoren nutzten jedoch die zuvor beschriebene Modellierung, in der anders als in der Abbildung Räume als Knoten und

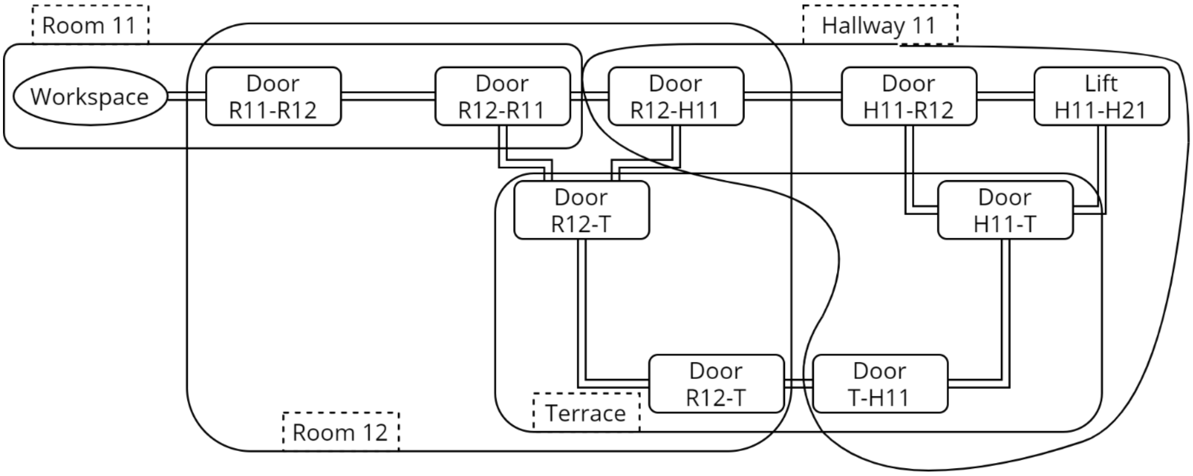


Abbildung 13 - Einfacher Hypergraph zur Darstellung von Durchgängen und Räumen

Durchgänge beziehungsweise Verbindungen als Kanten dargestellt sind. Der Hypergraph dient somit als Wissensrepräsentation der Gebäudetopologie. Dabei wird als Randbedingung vorausgesetzt, dass veränderliche Hindernisse wie zum Beispiel Möbel sowie statische Hindernisse in Räumen wie beispielsweise Säulen nicht mitbetrachtet werden (Palacz et al., 2019).

Die Erzeugung des Hypergraphen aus einer IFC-Datei folgt dabei dem Vorgehen von Ismail et al. (Ismail et al., 2017; Ismail et al., 2018). Zur Verarbeitung des Graphen wird die Abfragesprache *Cypher* genutzt, die Speicherung erfolgt als Neo4j-Modell. Die Ermittlung der Weglängen aus dem Graphen erfolgt durch Anwendung des A*-Algorithmus nach vorheriger Abbildung des Hypergraphen auf einen gewöhnlichen Graphen.

3.5.3. ANSATZ 3: HYBRIDES MODELL

Der rein topologische Ansatz aus Abschnitt 3.5.2 war für Siemiątkowska et al. (2013, S. 376) jedoch nicht ausreichend, da so wichtige geometrische Informationen verloren gehen. Dazu gehören Abstände und Anordnung von Durchgängen zwischen Räumen sowie die Dimensionen der Räume selbst. Daher kombinieren die Autoren Topologie und Geometrie in einem hierarchischen Graphen, wie er beispielhaft für ein Beispielgebäude (siehe Abbildung 11) in Abbildung 14 dargestellt ist. Dieser weist in vertikaler Richtung eine Baumstruktur auf, in der die Hierarchieebenen gegliedert sind. Die „Blätter" der untersten Hierarchieebene repräsentieren dabei aber nicht die Bezugsräume, sondern die Durchgänge zwischen diesen oder wichtige Orte innerhalb der Bezugsräume. Dazu zählen Türen und fiktive Bezugsraumgrenzen, aber auch Ladestationen, Arbeitsorte oder Lagerplätze. Allgemein repräsentiert also jeder Knoten einen Teil einer Hierarchie-Eben und jede Kante eine direkte Beziehung zwischen diesen („befindet sich in" oder „besteht aus"). In der horizontalen Ebene des Graphen werden Durchgänge nebeneinanderliegender Bezugsräume durch Kanten verbunden. Befindet sich beispielsweise *Raum 11* neben *Raum 12* und sind beide Räume durch eine Tür direkt verbunden, wird diese Tür als *Tür R11-R12*

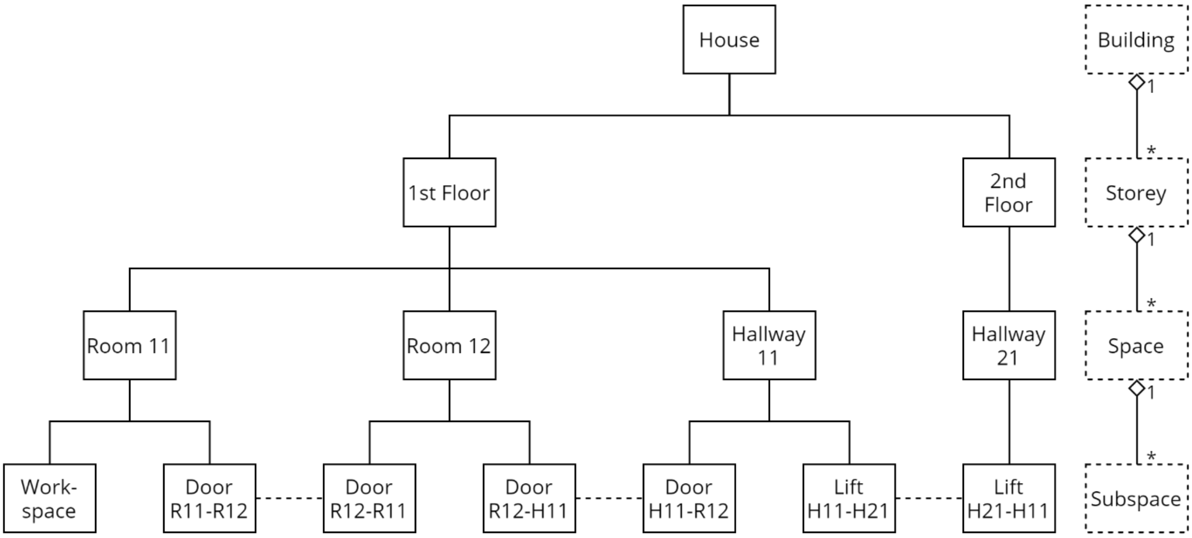


Abbildung 14 - Hierarchischer Graph für ein Beispielgebäude

des Raumes 11 mit *Tür R12-R11* des Raumes 12 verbunden, da es sich um denselben Durchgang handelt. Somit ist in horizontaler Ebene ersichtlich, welche Bezugsräume an welche gemeinsamen Durchgänge angrenzen (Siemiątkowska et al., 2013, S. 378).

Die Hierarchie selbst ergibt sich aus vier von den Autoren gewählten Hauptebenen eines Bauwerkes und ist an die in IFC definierte Gliederung eines Gebäudes angelehnt:

- Ebene 1: Gebäude (buildings)
- Ebene 2: Etagen (storeys)
- Ebene 3: Bereiche (spaces) – beispielsweise Räume oder Flure
- Ebene 4: Unterbereiche (subspaces) – beispielsweise Türen oder Arbeitsorte

Die Erzeugung eines solchen hierarchischen Graphen wurde durch Weiterverarbeitung eines Exportes aus der Software *Revit Architect* realisiert und nicht weiter spezifiziert. Dazu wurde auf eine bereits zuvor veröffentlichte Arbeit der Autoren Borkowski et al. (2010) zurückgegriffen (Borkowski et al., 2010, S. 724–734; Siemiątkowska et al., 2013, S. 376). Das Plugin Dynamo für Revit bietet aber beispielsweise Grundlagen einer solchen Funktionalität an.[12]

Navigation im hybriden (hierarchischen) Modell

Die Modellierung der Navigation erfolgt dann mithilfe von *Aktionen* und *Zuständen*, die wie folgt definiert werden: ein Aufenthalt an einem Knoten wird als örtlicher Zustand (state) des Roboters angesehen. Jede Änderung vom aktuellen Zustand zu einem neuen Zustand

---

[12] Siehe dazu auch knowledge.autodesk.com/search-result/caas/simplecontent/content/building-collaboration-data-extraction-bim-model.html (zuletzt abgerufen am 12.07.2021)

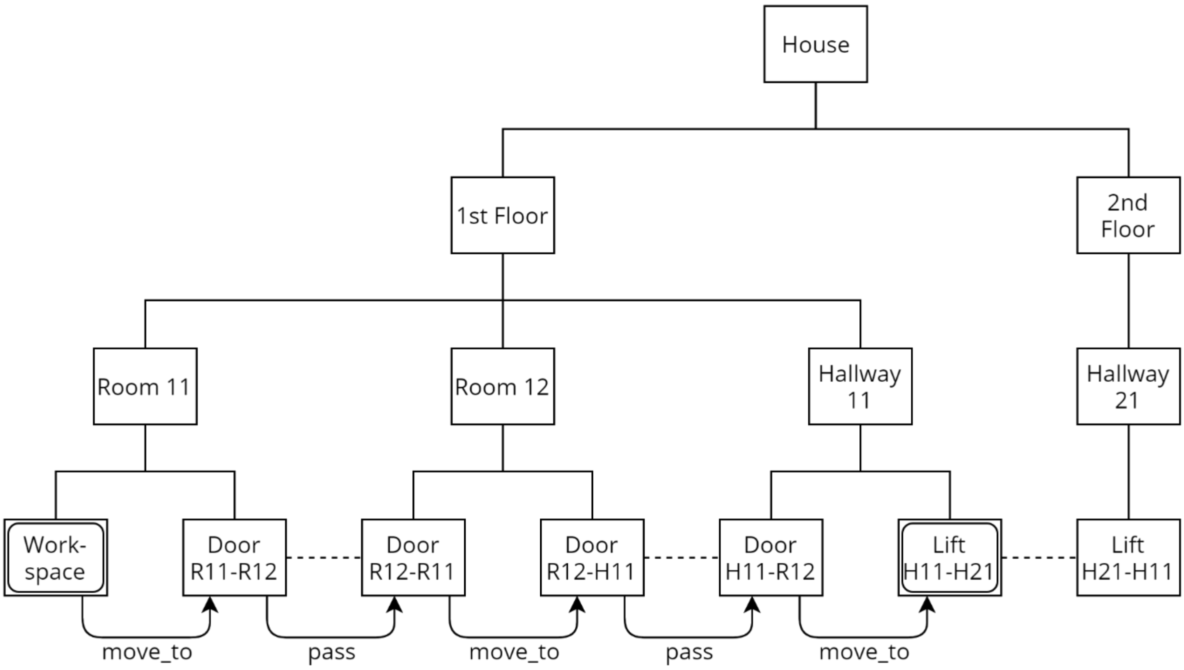


Abbildung 15 - Wegermittlung im hierarchischen Gebäudemodell

ist demgegenüber eine Abfolge einer oder mehrerer Aktionen (action). Diese Art der Modellierung durch Aktionen und Zustände entspricht einem Petri-Netz und kann dementsprechend auch mit den dafür geeigneten Algorithmen verarbeitet werden. Dabei gilt immer die grundlegende Annahme, dass man sich zwischen zwei Unterbereichen nur direkt bewegen kann, wenn sie durch dieselbe Tür verbunden sind oder im selben Element der darüberliegenden Hierarchieebene liegen, wie beispielsweise im selben Raum oder Flur. Indirekte Verbindungen zweier Unterbereiche lassen sich damit aus dem Durchlaufen des hierarchischen Graphen ermitteln. Dadurch ist auch eine Navigation über mehrere Etagen oder zwischen Räumen verschiedener Gebäude möglich. In Abbildung 15 ist ein solches Navigationsmodell zur Wegermittlung aus Aktionen und Zuständen dargestellt.

Die Kanten des Graphen lassen sich außerdem mit einer sogenannten Kostenfunktion belegen, welche die „Kosten“ zur Veränderung eines Zustands widerspiegeln. Das kann beispielsweise die benötigte Fahrzeit zwischen zwei Unterbereichen sein. Diese ergibt sich aus dem euklidischen Abstand der zwei betrachteten Unterbereiche und der möglichen Höchstgeschwindigkeit zwischen diesen. Die Kostenfunktion kann durchaus asymmetrisch sein, wodurch sich zum Beispiel Einbahnstraßen oder Steigungen modellieren lassen. Dadurch ist eine effiziente Suche nach dem schnellsten Weg möglich. Zusätzlich zur Fahrzeit zwischen zwei Knoten lassen sich auch die Knoten selbst mit einer Kostenfunktion belegen, um beispielsweise die Dauer zum Öffnen oder Schließen von Türen zu modellieren.

Je kleiner und zahlreicher die gewählten Unterbereiche sind, desto präziser ist dabei die Ermittlung der Fahrzeit. Ein Extremfall stellt die Rasterung der gesamten Grundfläche dar, um möglichst viele Orte im Gebäude präzise abbilden zu können. Dies kann jedoch bei der Berechnung des kürzesten Weges und beim Erstellen des hierarchischen Graphen zu hohem Speicher- und Rechenbedarf führen, was in der Anwendung in einer mobilen Roboterplattform unerwünscht sein kann.

Wegermittlung mithilfe eines Zellulären Neuronalen Netzwerkes
Die Ermittlung des kürzesten oder schnellsten Weges erfolgt dabei iterativ und in der Hierarchie von oben nach unten mithilfe eines *Zellulären Neuronalen Netzes*, kurz CNN[13] (Cellular Neural Network). Dies ist eine Sonderform eines Neuronalen Netzes, in dem Neuronen nur mit direkt benachbarten Neuronen verbunden sind. In Abbildung 16 ist ein solches CNN für die erste Etage des Beispielgebäudes dargestellt. Es ist um eine Terrasse erweitert, die ebenfalls Raum 12 und Flur 11 verbindet, um den Umgang mit mehreren möglichen Wegen zeigen zu können. Ein Neuron repräsentiert dabei einen Knoten beziehungsweise Unterbereich und die Verbindungen zwischen Neuronen die Dauer, die eine Zustandsänderung von einem Unterbereich zum nächsten benötigt. Die „Kosten" einer Zustandsänderung eines Neurons ergibt sich dann aus dem Produkt des Eingangssignals mit dessen Wichtung. Jedes Neuron ist dabei per Definition gezwungen, seinen eigenen Kostenwert zu minimieren und wählt entsprechend die Zustandsänderung, die für die aktuellen Eingangssignale die geringsten Kosten verursacht. Die Kosten der Aktion eines Neurons sind gleichzeitig dessen Ausgangswert an benachbarte Neuronen. Daher verursacht jede Zustandsänderung eines Neurons gleichzeitig eine Anpassung der benachbarten Neuronen, da sich für diese wiederum möglicherweise neue günstige Zustandsänderungen ergeben. So könnte sich beispielsweise ein zuerst ermittelter, schnellster Weg von Raum 12 zu Flur 11 als ungünstig ergeben und durch einen Weg über die Terrasse ersetzt werden, weil das Öffnen der Tür von Raum 12 zu Flur 11 besonders lange dauert. Dies geschieht solange im gesamten CNN, bis es zu einer Konvergenz kommt und somit ein

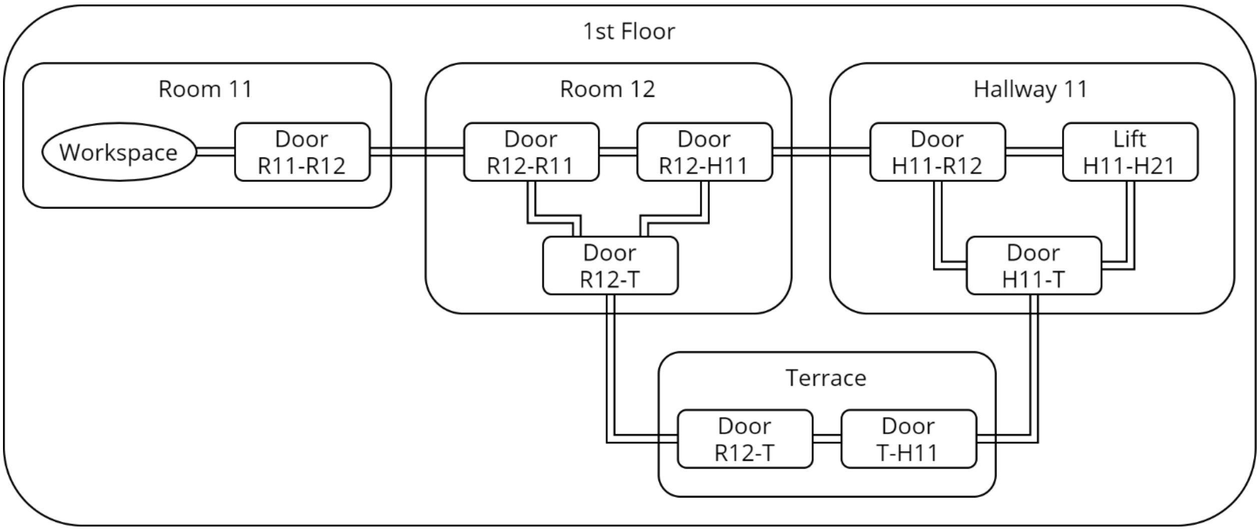


Abbildung 16 - Repräsentation des hybriden Modells durch ein CNN

---

[13] Nicht zu verwechseln mit *Convolutional Neural Networks*, die ebenfalls als CNN abgekürzt werden.

stabiler Zustand eintritt. Das bedeutet, dass sich kein Neuron mehr für einen neuen, günstigeren Zustand entscheidet und keine Veränderungen mehr stattfinden. Das Ergebnis der Wegfindung ist daher eine Abfolge aus Aktionen und Zuständen, die nacheinander ausgeführt beziehungsweise erreicht werden (Przybylski & Siemiątkowska, 2012, S. 485; Siemiątkowska et al., 2013, S. 377–380).

Semantische Erkennung

Damit der Roboter den aktuell besuchten Raum semantisch nach Nutzungsart erkennen kann, wird zusätzlich eine Bilderkennung genutzt. Dazu werden zwei Informationsquellen am Roboter untersucht und kombiniert. Zum einen wird eine optische Erkennung von flachen, zweidimensionalen Objekten wie beispielsweise Schilder mit Raumnummern vorgenommen. Dies erfolgt durch zwei an den Seiten des Roboters angebrachte Farbkameras. Mithilfe der offenen Programmbibliothek *openCV* werden einige typische Formen von Schildern wie beispielsweise Kreise oder Rechtecke aus den aufgenommen Bildern erkannt. Sobald ein Schild erkannt wird, extrahiert die Software die Beschriftung des Schildes, sodass die enthaltenen Informationen zusätzlich zur Lokalisierung genutzt werden können.

Zum anderen nimmt eine „3D-Kamera“ dauerhaft dreidimensionale Punktwolken auf, aus denen bekannte Objekte erkannt werden können. So werden unter anderem Türen, Möbelstücke oder Fenster erkannt. Voraussetzung dafür ist ein zuvor angelerntes Künstliches Neuronales Netz, was auf erwartete Objekte trainiert werden muss. Jeder Raumnutzungstyp wird im topologischen Modell mit einer Menge zu erwartenden Objekten beschrieben. So wird in einem Einzelbüro mindestens ein Stuhl, ein Schreibtisch und ein Schrank an der Wand erwartet, während in einem Badezimmer Waschbecken und Toilettenarmaturen erwartet werden. Je nach erkannten Objekten im Raum wird eine Wahrscheinlichkeit ermittelt, in welchem Ort des topologischen Modells sich der Roboter gerade befindet. Ab einem bestimmten Schwellwert der Wahrscheinlichkeit wird eine Position in einem konkreten (Unter-) Bereich angenommen (Siemiątkowska et al., 2013, S. 380–381).

Der vorgestellte Ansatz basiert jedoch maßgeblich auf dem Vorhandensein von Inneneinrichtung, was während der Rohbauphase jedoch nicht der Fall ist. Als Planungshilfe für den Fahrweg des Roboters ist die Nutzung eines hybriden Modells für präzise Weg- und Zeitermittlungen aber vielversprechend. Schon grundlegende Erkennung der Türen- und Fensteranzahl im Sichtbereich des Roboters in Kombination mit dem Wissen aus dem BIM-Modell kann zu besserer Lokalisierungsfähigkeit führen.

3.6. SLAM – SIMULTANEOUS LOCALIZATION AND MAPPING

Die Methode des gleichzeitigen Kartierens und Lokalisierens oder kurz SLAM (Simultaneous Localization and Mapping) ermöglicht es, in komplett oder teilweise unbekannten Umgebungen autonom navigieren zu können. Formell lässt sich SLAM folgendermaßen beschreiben: ein Roboter befährt eine noch unbekannte, ebene Umgebung. Dessen Position x_t zu einem Zeitpunkt t lässt sich durch eine zweidimensionale Koordinate und dem Winkel der Rotation des Roboters relativ zum Koordinatensystem der Ebene beschreiben. Die zeitliche Abfolge der Positionen des Roboters lässt sich zusammenfassen zu

$$X_T = \{x_0, x_1, x_2, \dots, x_T\} \tag{3.10}$$

und ist nicht direkt bekannt. Um aus einem bekannten Startpunkt einen neuen Punkt zu ermitteln, wird die relative Bewegung des Roboters zur Ebene durch Odometrie ermittelt und daraus die neue Position berechnet. Dies kann beispielsweise durch Zählen der Radumdrehungen und Messen des Lenkwinkels umgesetzt werden. Es ergibt sich eine Abfolge von Odometriemessungen

$$U_T = \{u_1, u_2, \dots, u_T\} \tag{3.11}$$

wobei eine Messung u_t die relative Bewegung zwischen einem Zeitpunkt t und $t-1$ darstellt. Weiterhin wird angenommen, dass eine Karte der tatsächlichen Umgebung in einem Datenmodell m gespeichert wird. Das heißt, dass die Position von Landmarken, Objekten oder Oberflächen in der Umgebung aufgenommen wird. Dies wird beispielsweise durch Tiefenmessungen mit LIDAR-Systemen durch den Roboter umgesetzt. Eine solche Messung z_t lässt sich als Abfolge von Messungen zusammenfassen:

$$Z_T = \{z_1, z_2, \dots, z_T\} \tag{3.12}$$

Ziel des SLAM-Algorithmus ist es nun, anhand der Messung der Umgebungseigenschaften Z_T und der Odometrie U_T des Roboters die Karte m der tatsächlichen Umgebung und die Abfolge der Roboterpositionen X_T in m zu rekonstruieren. Die Rekonstruktion der gesamten Bewegung des Roboters erfolgt dabei probabilistisch zu

$$p(X_T, m | Z_T, U_T) \tag{3.13}$$

und wird auch *full SLAM* genannt. Analog dazu kann das Problem des *online SLAM* formuliert werden, bei dem anstatt aller Zeitpunkte auf einmal nur die Position zum Zeitpunkt $t \in T$ ermittelt wird:

$$p(X_t, m | Z_t, U_t). \tag{3.14}$$

Für die tatsächliche Lösung des SLAM-Problems müssen zusätzlich noch zwei Abbildungsvorschriften mathematisch formuliert werden:

$$(u_t, x_{t-1}) \mapsto x_t(u_t, x_{t-1}) \tag{3.15}$$

zur Ermittlung der erwarteten aktuellen Position aus der vorherigen Position und der Odometrie des Roboters sowie

$$(x_t, m) \mapsto Z_t(x_t, m) \tag{3.16}$$

um zu formulieren, von welcher Position aus der Roboter welche Eigenschaften der Karte messen kann. Dies wird ebenfalls wie in den Gleichungen (3.13) und (3.14) als Wahrscheinlichkeitsverteilung aufgefasst, da alle Variablen der Abbildungen unsicher sind (Stachniss et al., 2016).

Je nach konkreter Abbildungsvorschrift werden drei grundsätzliche Techniken des SLAM-Algorithmus unterschieden: Extended Kalman Filter (EKF), Partikelfilter und graphenbasierte Techniken. Die in dieser Diplomarbeit verwendete Software *nav2* nutzt die von Macenski (2019) vorgestellte *SLAM Toolbox*. Diese nutzt Partikelfilter und graphenbasierte Ansätze (Macenski, 2019; Macenski, Martín et al., 2020), weshalb diese im Folgenden kurz vorgestellt werden.

3.6.1. PARTIKELFILTER

Partikelfilter repräsentieren eine *A-posteriori-Wahrscheinlichkeit*[14] durch eine Menge von fiktiven, gewichteten *Partikeln*. Eine solche im Englischen auch kurz als *posterior* („im Nachhinein“) bezeichnete Wahrscheinlichkeit beschreibt im Allgemeinen Vermutungen über einen unbekannten Umweltzustand anhand bisher gemachter Beobachtungen einer Zufallsgröße. Dabei ist der unbekannte Umweltzustand von der beobachtbaren Zufallsgröße statistisch abhängig, wodurch sich Schlüsse auf den Umweltzustand ziehen lassen. Ein einzelner Partikel lässt sich somit als eine konkrete Vermutung ansehen, in welchem Zustand sich die Umgebung des Roboters soeben befindet. Durch eine Vielzahl von Partikeln ergibt sich eine Annäherung an die unbekannte Wahrscheinlichkeitsverteilung. Ein Partikelfilter lässt sich somit auch als nichtparametrische Repräsentation einer multimodalen Wahrscheinlichkeitsverteilung ansehen. Multimodal meint in diesem Kontext, dass die Wahrscheinlichkeitsverteilung mehrere lokale Maxima aufweist. Dies ergibt sich daraus, dass beispielsweise mehrere mögliche Umgebungszustände ähnlich wahrscheinlich sein können. Bei identischen Raumgrundrissen ist zum Beispiel zwar sofort erkennbar, wo im Raum selbst sich der Roboter befindet, aber nicht, in welchem Raum.

Um eine Näherung der zugrundeliegenden Wahrscheinlichkeitsverteilung aus mehreren Partikeln ableiten zu können, werden alle Partikel mit einer Wertung beziehungsweise

---

[14] Es handelt sich hierbei um einen Begriff aus dem Gebiet der bayesschen Statistik, dessen genauere Definition den Rahmen der Arbeit übersteigt.

einem Gewicht versehen. Diese Wichtung wird nach jeder Messung aktualisiert, um neue Erkenntnisse in die Kartierung einfließen zu lassen.

Partikelfilter für SLAM-Probleme weisen allerdings zwei signifikante Nachteile auf. Zum einen ist die Anzahl der benötigten Partikel für eine ausreichend genaue Abdeckung der Umgebung unbekannt und kann nur durch auf Sachkenntnis gestützte Vermutungen oder vorherige Tests festgelegt werden. Je nach Grad der Unbekanntheit der Umgebung kann dies sehr schwierig sein und stellt auch immer eine Abwägung aus Rechenleistung und Genauigkeit dar. Zum anderen kann häufiges, mehrfaches Besuchen bereits kartierter Gebiete die Erzeugung einer konsistenten Karte verhindern, da dann wenige Partikel stark und die verbleibenden Partikel kaum gewichtet werden. Partikelfilter lassen sich jedoch gut für Anwendungen des *online SLAM* nutzen, weshalb sie dennoch eine große Bedeutung haben (Durrant-Whyte & Henderson, 2016; Stachniss et al., 2016; Thrun, 2002).

3.6.2. GRAPHENBASIERTES SLAM

Das SLAM-Problem lässt sich auch durch Beschreibung der Zustandsvariablen in einem Graph lösen. Dazu werden die Positionen des Roboters sowie signifikante Orte wie beispielsweise Flächen oder Ecken der Umgebung als Knoten eines Graphen aufgefasst. Kanten im Graph dagegen stellen Randbedingungen dar, die anhand von Odometrie und Messung der Umgebungseigenschaften erhalten werden. Kanten werden anhand der Informationen aus Odometrie zwischen aufeinanderfolgenden Positionen (x_{t-1}, x_t) des Roboters gezogen. Weiterhin werden Kanten zwischen einer Position x_t und allen von dieser Position aus beobachteten signifikanten Orten m_i der Umgebung erzeugt. Mit jeder Bewegung des Roboters wächst der Graph weiter an, bleibt bei beschränkter Reichweite der Sensoren jedoch dünn besiedelt. Das heißt, dass ein Knoten nur mit wenigen anderen Knoten verbunden ist, auch wenn insgesamt viele Knoten existieren. Daher lassen sich nach einer Speicherung des Graphen als Matrix effiziente, nichtlineare Optimierungsalgorithmen für dünn besiedelte Matrizen anwenden. Die Optimierung selbst stellt dabei die Lösung des SLAM-Problems dar und wird durch Entspannung der Randbedingungen erhalten: betrachtet man die Kanten des Graphs als gespannte Federn, ist der optimale Zustand des Systems der, in welchem am wenigsten Spannung im System herrscht. Dies ist nach dem Prinzip der minimalen Energie der wahrscheinlichste Zustand des Systems. Graphenbasierte SLAM-Algorithmen sind auch für sehr große Karten beziehungsweise Gebiete skalierbar, weshalb immer mehr Verbesserungen und Implementierungen für diese entwickelt werden (Stachniss et al., 2016).

Informationen über die Umgebung kann ein Roboter allerdings nur eingeschränkt während seines Einsatzes selbstständig erfassen. Nicht-physikalische oder zumindest nicht direkt aus physischen Eigenschaften ableitbare Informationen wie Semantik und Topologie einer Umgebung sind jedoch für einen Roboter (noch) nicht selbst erfassbar. Auch die physische Beschaffenheit der Umgebung lässt sich in seiner Gesamtheit nur aufwendig durch Abfahren und Analysieren aller Orte durch den Roboter umsetzen. Dabei kommt es zusätzlich noch zu typischen Verzerrungen der erzeugten Karte, da sich ein Messfehler, je weiter die Messung vom Ausgangspunkt getätigt wird, immer stärker auswirkt. Eigentlich rechtwinklige Gebäude werden im Ergebnis meist bananenförmig abgebildet. Die Nutzung von bereits vorhandenen Informationen aus dem BIM-Modell für Kartenmaterial ist daher unumgänglich (Nitta et al., 2020, S. 822–825).

3.6.3. AMCL – ADAPTIVE MONTE CARLO LOCALIZATION

Ein eng verwandtes Problem zu SLAM beziehungsweise ein Teilproblem dessen ist das der globalen Lokalisierung innerhalb einer bereits bekannten Karte, selbst wenn diese die Umgebung nicht exakt abbildet. Grundlage für den in *nav2* genutzten AMCL-Algorithmus ist die Arbeit von Fox et al. (1999), welche wie bei Partikelfiltern (siehe Abschnitt 3.6) die Nutzung von Partikeln vorschlägt. Ein solcher Partikel entspricht analog dem SLAM-Algorithmus ebenfalls einer Vermutung, wo sich der Roboter derzeit auf der Karte befindet. Um im Lösungsraum der möglichen Positionen des Roboters schnell wahrscheinliche Lösungen ermitteln zu können, wird die sogenannte *Adaptive Monte Carlo Localization* angewandt. Das Vorgehen entspricht dem der Partikelfilter des SLAM-Problems und wird durch die zusätzliche Verwendung einer Anzahl immer wieder zufällig neu generierter Partikel erweitert. Dadurch wird sichergestellt, dass auch eine irrtümlich angenommene Position auf der Karte noch korrigiert werden kann, selbst wenn diese zuerst weit von der tatsächlichen Position entfernt war (Durrant-Whyte & Henderson, 2016).

3.7. PRAKTISCHE UMSETZUNG DER INFORMATIONSÜBERGABE AM EINSATZORT

Prinzipiell kommen zwei Arten der Informationsübergabe in Frage: die kabellose Übertragung über ein geeignetes Funknetzwerk und die kabelgebundene Übertragung über eine angedockte Basisstation oder unter Verwendung eines tragbaren Speichermediums. Außerdem muss betrachtet werden, wie die Übertragung vom zentralen BIM-Modell zur verwendeten Übertragungstechnologie zum Roboter erfolgt.

Wie in Abschnitt 3.2.5 bereits aufgezeigt, ist die Verwendung kabelloser Technologien mithilfe von Funknetzwerken auf Baustellen mit einigen besonderen Herausforderungen verbunden. Die häufig im Hochbau verwendete Stahlbetonbauweise schirmt elektromagnetische Strahlung ab, was die Übertragungsgeschwindigkeit von Daten stark vermindern oder ganz unmöglich machen kann. Weiterhin muss entweder auf bereits vorhandene Funknetzwerke wie dem Mobilfunk zurückgegriffen werden, oder ein eigenes, lokales Funknetzwerk aufgebaut werden. Der Empfang von Mobilfunksignalen stellt sich jedoch lokal in schlecht erschlossenen Gebieten oft als unzuverlässig und langsam heraus, was auch die Karte der Bundesnetzagentur[15] zeigt. In persönlichen Gesprächen mit mehreren in der Baupraxis tätigen Personen und aus persönlicher Erfahrung bestätigt sich diese Erkenntnis zusätzlich. Zur mobilfunkbasierten Steuerung von Robotern beziehungsweise Datenübertragung existieren zwar umfangreiche Lösungen, deren Zuverlässigkeit aber stark vom Mobilfunkempfang abhängt (Kadena et al., 2021; Kahar et al., 2011). Auch die Sicherheit von direkt mit dem Internet verbundenen Geräten jeder Art muss hier besonders beachtet werden, um Missbrauch und Datendiebstahl vorzubeugen.

Lokale kabellose Technologien wie WLAN oder Breitbandfunk (UWB) benötigen eine Infrastruktur, die erst aufwendig vor Ort aufgebaut und mit Strom versorgt werden muss. Dies ist auf Baustellen im Regelfall nicht durchführbar. Eine Netzwerkversorgung nur an bestimmten Orten des Gebäudes wäre als Kompromisslösung denkbar. So könnte an jedem (Haupt-) Stromverteiler einer Etage ein WLAN-Router miteingebaut werden. Der Roboter müsste dann zum Datenaustausch in die Nähe des Routers fahren und dort den Übertragungsvorgang abwarten. Die Verkabelung zwischen den einzelnen Stromverteilern mit Netzwerktechnik stellt jedoch selbst wieder eine Schwierigkeit dar, die weitere Untersuchungen benötigt.

Da ein akkubetriebener Roboter aber in jedem Fall auch eine Ladestation benötigt, die ebenfalls mit Strom versorgt werden muss, ist die Datenübergabe direkt an der Ladestation in Betracht zu ziehen. Im Baustellenbetrieb wird der Roboter vor Arbeitsbeginn immer an dessen Ladestation stehen. In diese kann beispielsweise ein Bluetoothmodul integriert werden, über das der Datenaustausch erfolgt (Kahar et al., 2011). Dennoch muss auch hier eine Verbindung zwischen den Daten des BIM-Modells und der Ladestation hergestellt werden. Dies muss wiederum via Netzwerkkabel oder Funknetzwerk erfolgen.

Eine pragmatische Lösung kann die manuelle Übertragung über einen tragbaren Datenträger oder ein Smartphone beziehungsweise Tablet sein. Im Baucontainer beziehungsweise temporären Baustellenbüro der Bauleitung ist fast immer ein Computer mit Netzwerkanschluss aufgebaut, sodass hier auch ein mobiles Endgerät mit Daten versorgt werden kann, ohne auf der Baustelle selbst auf ein Funknetzwerk angewiesen zu sein. Es muss dann allerdings händisch das Gerät zum Roboter getragen und der Datentransfer angewiesen werden, beispielsweise über USB, Bluetooth oder WLAN. Da sich jedoch auch die gesamte Robotersteuerung über ein Tablet oder Smartphone abwickeln lässt (Kadena et al., 2021) und insbesondere Tablets auch schon in anderen Bereichen auf Baustellen zum Einsatz kommen, könnte sich diese Lösung in der praktischen Umsetzung am einfachsten und sichersten gestalten.

---

[15] Online abrufbar unter breitband-monitor.de/mobilfunkmonitoring/karte (Zuletzt abgerufen am 14.09.2021)

4 IMPLEMENTIERUNG EINES BEISPIELS

Die Implementierung des Beispiels besteht aus zwei grundlegenden Teilen. Zum einen muss aus der IFC-Datei eine Karte sowie eine Simulationsumgebung erzeugt werden. Zum anderen muss die Karte anschließend durch einen Roboter mit entsprechender Sensorik in der Simulationsumgebung genutzt werden können. Zuvor müssen noch angemessene Randbedingungen für die praktische Umsetzung definiert werden.

4.1. RANDBEDINGUNGEN

In den vorherigen Kapiteln wurde bereits auf die Diversität von Organisation, Aufbau und Ablauf im Rahmen einer Baustelle hingewiesen. Daher kann es zum derzeitigen Stand keine allgemeingültige Lösung für alle Bauprojekte geben. Im Rahmen der gleichzeitig mit dieser Diplomarbeit laufenden Abschlussarbeiten am Institut für Bauinformatik wird als Ziel des Beispiels ein Roboter simuliert, der in der Lage sein soll, überwiegend selbstständig Tätigkeiten des Innenausbaus wie zum Beispiel Malerarbeiten ausführen können. Dazu soll er von seinem aktuellen Standort zum gewünschten Arbeits- beziehungsweise Einsatzort navigieren können. Aufgrund der Vielzahl an Faktoren, die die zuvor genannte Punkte beeinflussen, werden folgende Festlegungen für das umgesetzte Beispiel getroffen:

- Für das Gebäude ist ein BIM-Modell vorhanden, in dem die Geometrie auf wenige Zentimeter genau modelliert ist. Wie in Abschnitt 2.3.1 erläutert, wird ein Leistungsniveau 2 für das BIM-Modell vorausgesetzt.
- Der Baufortschritt ist bereits soweit fortgeschritten, dass auf den zu befahrenden Böden ein Estrich eingebaut und voll belastbar ist. Der Boden ist daher eben und ein Durchdrehen der Räder unwahrscheinlich.
- Das Gebäude hat barrierefreie oder zumindest barrierearme Raumübergänge. Es sind also alle Durchgänge beziehungsweise Türen breit genug, um den Roboter problemlos hindurchfahren zu können. Türschwellen sind nur wenige Zentimeter hoch und lassen sich daher vom Roboter überfahren.
- Der Roboter inklusive Aufbauten und eventuell transportiertem Material überschreitet die Belastbarkeit des Estrichs und des Fußbodens allgemein nicht.

Die Randbedingungen wurden im Rahmen einer Baustellenbesichtigung eines Wohnkomplexes diskutiert und festgelegt. Sie sollen eine Balance aus Umsetzbarkeit und praktischer Relevanz des Beispiels ermöglichen.

4.2. GENUTZTE SOFT- UND HARDWARE

Da für die Diplomarbeit eine Reihe teilweise voneinander abhängiger Soft- und Hardware verwendet wurde, wird diese hier detailliert aufgelistet, um Ergebnisse nachvollziehbar zu machen. Mit Ausnahme des Programms *FZKViewer* wurde ausschließlich mit dem Betriebssystem *Ubuntu* gearbeitet.

4.2.1. ROBOTERMODELL

Zusätzlich zu den Softwarepaketen wurde maßgeblich auf die freie und quelloffene Implementierung des Roboters *TurtleBot3* der Firma *Robotis, Inc.* zurückgegriffen. Das Modell *TurtleBot3 Waffle Pi* ist ein frei verkäuflicher mobiler Roboter, der unter anderem mit einer Kamera und einem LIDAR-Modul ausgestattet ist. Die circa 30 mal 30 Zentimeter große mobile Plattform ist in ihren Funktionen umfangreich dokumentiert und bietet fertige Pakete für ROS und ROS 2 an, die ein zügiges Erzeugen von lauffähigen Simulationen sowie realen Tests erlauben (Robotis, Inc., 2021b). Für einen realen Einsatz auf einer Baustelle ist diese Plattform zwar zu klein und daher nicht geeignet, soll hier aber als Simulationsgrundlage ausreichend sein, um grundlegende Konzepte zu verdeutlichen und zu testen.

Der TurtleBot3 ist mit dem 360-Grad-LIDAR *LDS-01* ausgestattet. Es hat in der Realität eine effektive Reichweite von 3,5 Metern (Robotis, Inc., 2021a). Für die Simulation wurde diese Reichweite künstlich auf 15 Meter erhöht, was als Simulationsparameter problemlos möglich ist. Für einen praktischen Einsatz wäre das originale LIDAR-System des TurtleBot3 in seiner Reichweite viel zu klein. Die Anpassung erfolgt in der Konfigurationsdatei des Roboters in `models\turtlebot3_waffle\model.sdf` im Feld `<range>`.

4.2.2. SOFTWARE

Betrachtung, Konvertierung und Verarbeitung von IFC-Dateien:

- FZKViewer, Version 6.1 Build 1816 (Karlsruhe Institute of Technology [KIT], 2020)
- IfcOpenShell, Version 0.6.0b0 (OCC 7.3.0) (Krijnen, 2021b) mit Open Cascade (Open Cascade S.A.S, 2020)
- IfcConvert, Version 0.6.0 (Krijnen, 2021a)
- Wrapper für IfcOpenShell in Python: PythonOCC, Version 7.5.1 (Paviot, 2021)

Bearbeitung des dreidimensionalen Simulationsmodells:

- Blender, Version 2.82.7 (Blender Institute, 2021)
- MeshLab, Version v2020.03 (Cignoni et al., 2020)

Simulation des Roboters:

- ROS 2 Foxy Fitzroy (Open Robotics, 2020c) inklusive der Programme rviz und Gazebo, Version 11.5.1
- ROS Navigationspaket nav2 (Macenski, Martín et al., 2020) sowie eigenständig modifizierte Version des Teilpaketes nav2_bringup, Version 0.4.7 (Macenski et al., 2021)

Kartenerstellung:

- Eigenes Python-Skript nach Sears-Collins und Sperbeck (Sears-Collins & Sperbeck, 2021)
- Konvertierung als PGM mit ImageMagick, Version 6.9.10-23 (ImageMagick Studio LLC, 2021)

4.2.3. HARDWARE

Das für alle Arbeitsschritte eingesetzte System hat folgende Spezifikationen:

- Betriebssystem: Ubuntu 20.04.2.0 LTS; Windows 10 Pro
- Prozessor: Intel Core i7-7600U CPU @ 2.80GHz
- Graphik: Intel HD Graphics 620 und NVIDIA Quadro M520 mit proprietärem NVIDIA Graphiktreiber, Version 470.63.01
- Arbeitsspeicher: 32 GB SO-DIMM DDR 4 @ 2400MHz
- SSD: Samsung SM961 m.2

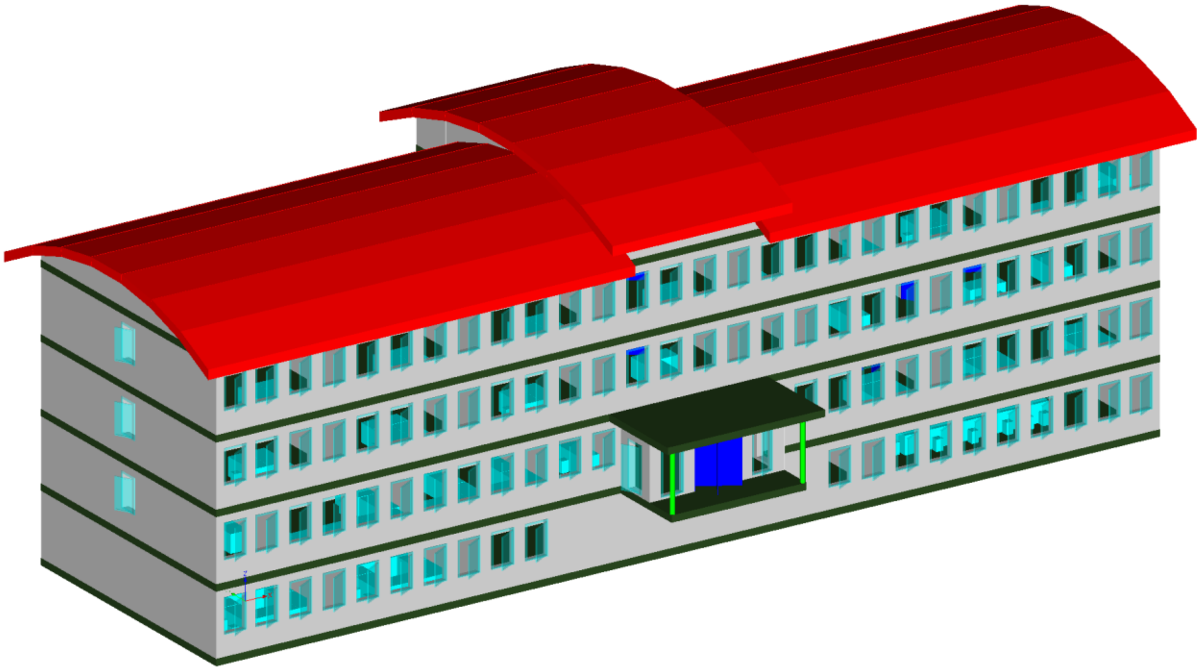


Abbildung 17 - Frontalansicht Bürogebäude (Screenshot FZK-Viewer)

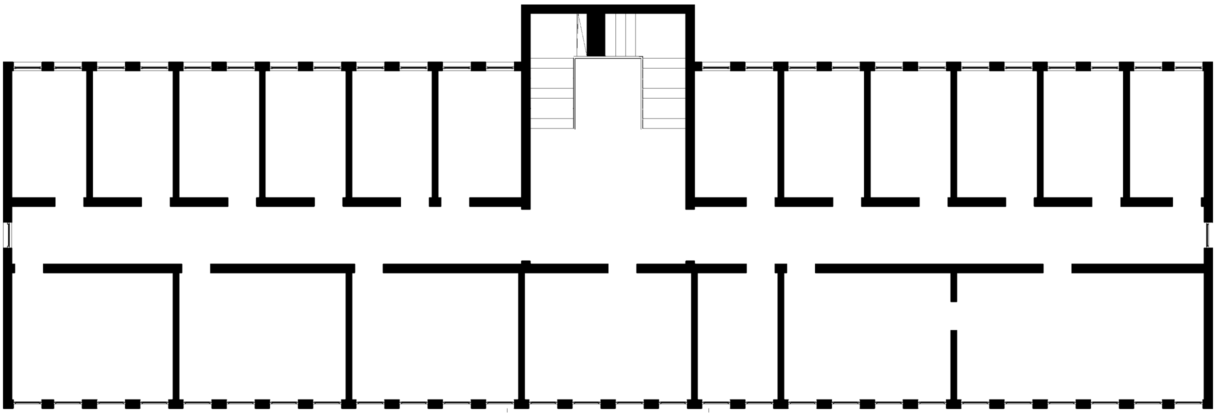


Abbildung 18 - Grundriss der ersten Etage

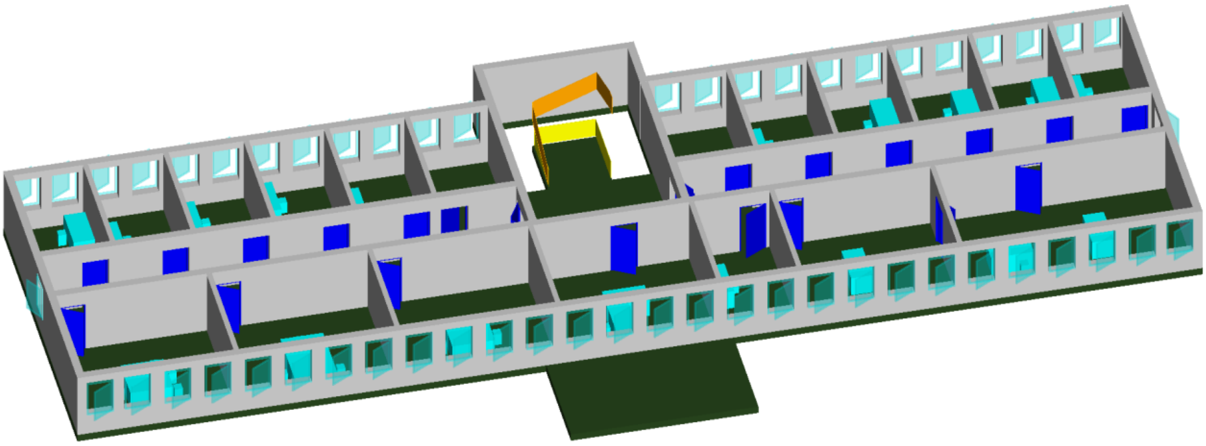


Abbildung 19 - Erste Etage des Bürogebäudes (Screenshot FZK-Viewer)

4.3. VORSTELLUNG DER BEISPIELGEBÄUDE

Für die Simulation der Implementierung wurden zwei Beispielgebäude ausgewählt. Diese werden im Folgenden kurz vorgestellt und wichtige Eigenschaften erläutert. Beide Gebäude liegen als IFC-Datei beziehungsweise STEP-File vor.

4.3.1. BEISPIELGEBÄUDE 1: FIKTIVES BÜROGEBÄUDE DES KIT

Das genutzte Beispielgebäude 1 ist ein frei verfügbares IFC-Modell eines fiktiven Bürogebäudes. Es wird vom Karlsruher Institut für Technologie (KIT) bereitgestellt und ist als dreidimensionales Rendering in Abbildung 17 dargestellt (KIT, 2021). Die verwendete IFC-Version ist Version 4. Das Gebäude besteht aus fünf Etagen, wobei die fünfte Etage als Dachgeschoss kein Vollgeschoss ist und die erste Etage als Kellergeschoss unter der Erdoberfläche liegt. Die IFC-Datei enthält insgesamt 1191 IFC-Entitäten und 6374 Relationen[16]. Die

---

[16] Angaben lt. *FZK-Viewer* des Karlsruhe Institute of Technology (2020).

Simulationsumgebung wurde aus der ersten Etage erstellt, die in Abbildung 19 als dreidimensionales Modell von schräg oben und in Abbildung 18 im Grundriss abgebildet ist.

4.3.2. BEISPIELGEBÄUDE 2: WOHNKOMPLEX

Das zweite Beispielgebäude ist ein real existierender Wohnkomplex in Leipzig, der freundlicherweise von der Firma *Ed. Züblin AG* zur Verfügung gestellt wurde, jedoch nicht zur weiteren Veröffentlichung freigegeben ist. Das Gebäude besteht aus 9 Etagen, von denen zwei unterirdisch liegen. Die IFC-Datei enthält insgesamt 22263 IFC-Entitäten und 55699 Relationen[17] und ist damit deutlich umfangreicher als die des Beispielgebäudes 1. Die IFC-Version des Modells ist Version 2x3. Die Simulationsumgebung wurde aus der ersten Etage erzeugt, die aus 3423 IFC-Entitäten besteht[18].

4.4. NAVIGATIONSMODELL AUS IFC-DATEI

Bei der Implementierung des Beispiels wurde der Fokus auf die Erzeugung einer nutzbaren Karte direkt aus der IFC-Datei gelegt. Prinzipiell bieten auch kommerzielle Softwareprodukte wie zum Beispiel *Autodesk Revit* eigene Exportfunktionen der Geometriedaten und teilweise auch von Modellen mit semantischen Informationen an. Die exportierten Dateien liegen dann jedoch meist in proprietären Formaten vor und sind auf die Softwarefamilie des Herstellers beschränkt. Die Verwendung von Software verschiedener Hersteller wird dadurch erschwert oder sogar unmöglich und es müssen Lizenzen für alle genutzten Programme vorhanden sein. Die Verwendung proprietärer Austauschformate widerspricht damit auch dem openBIM-Ansatz.

Die Karte wird für eine Etage eines Gebäudes als zweidimensionaler Grundriss erzeugt und als Bilddatei gespeichert. Anschließend wird der Maßstab der Karte ermittelt und die Bilddatei entsprechend in Höhe und Breite skaliert. Dadurch bleibt die Karte einerseits

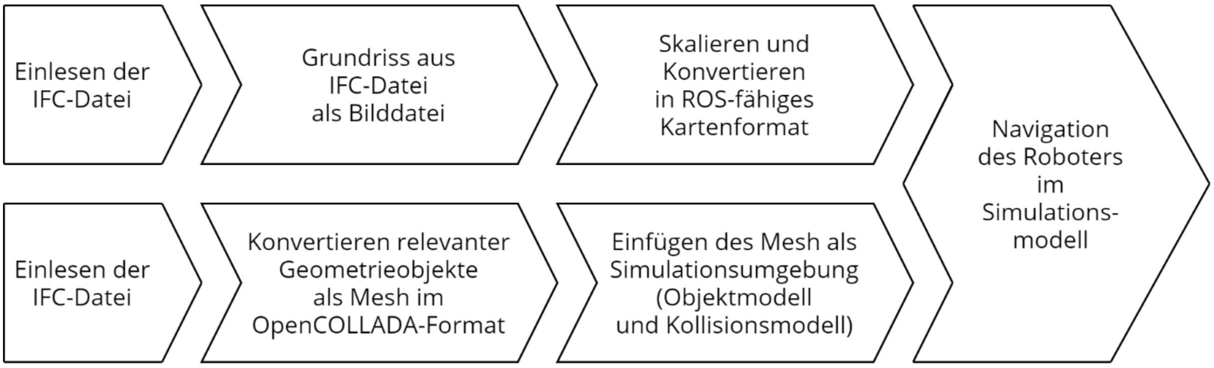


Abbildung 20 - Ablaufplan zur Erzeugung der ROS-fähigen Karte sowie der Simulationsumgebung für Gazebo

---

[17] Angaben lt. *FZK-Viewer* des KIT (2020).

[18] Angaben lt. *FZK-Viewer* des KIT (2020).

detailliert genug, um sie als Navigationsgrundlage nutzen können und andererseits ausreichend komprimiert, um wenig Arbeitsspeicherbedarf beim Navigieren zu erfordern.

Weiterhin werden aus der IFC-Datei für die Simulation relevante Objekte des Gebäudes in ein sogenanntes *Mesh* umgewandelt. Es handelt sich dabei um eine direkte Repräsentation dreidimensionaler Objekte durch Vielecke. Dieses Mesh bildet die Grundlage für die graphische Darstellung der Simulationsumgebung sowie als physikalische Simulation der Umgebung für die Abtastung durch Sensoren und Kollisionserkennung. Die Erzeugung des Meshs und der Simulationsumgebung allgemein wird in Abschnitt 4.5 genauer erläutert. In Abbildung 20 ist der prinzipielle Ablauf der praktischen Umsetzung für die Erstellung der Karte und der Simulationsumgebung dargestellt. Im Anschluss an diese vorbereitenden Schritte kann die Simulation gestartet werden.

4.4.1. GRUNDRISSGEOMETRIE AUS IFC-DATEI

Die Erzeugung der Grundrissgeometrie aus der IFC-Datei folgt dem Vorschlag von Follini et al. (2021). Grundsätzlich wird dazu aus einem horizontalen Schnitt durch das Gebäudemodell eine Bilddatei erzeugt, die anschließend für ROS optimiert wird und als zweidimensionale Karte dient. Es werden alle Elemente, die von der Elternklasse *IfcProdukt* erben und eine geometrische Repräsentation haben, in ein neues Geometriemodell überführt. Dieses wird auf der Kamerahöhe des Roboters horizontal geschnitten. Die Höhe muss derzeit noch händisch angegeben werden.

Realisiert wird dies durch das Programmpaket *IfcOpenShell* (Krijnen, 2021b). Es ermöglicht eine detailliertere Vorfilterung der IFC-Datei wie beispielsweise nach Materialtypen oder Bauteilhöhe sowie das Erzeugen von Schnitten durch eine Ebene direkt aus der IFC-Datei. Derzeit ist allerdings in der Software *OpenCascade*, die von *IfcOpenShell* als Geometrie-Bibliothek[19] zur Umsetzung der geometrischen Repräsentationen aus der IFC-Datei genutzt wird, ein Fehler enthalten. Dieser führt unter anderem bei Schnittoperationen von Öffnungen aus Elementen, wie sie beispielsweise bei *IfcOpeningElement* nötig sind, zu einem Absturz von *OpenCascade*. Der Fehler ist dokumentiert[20] und mindestens seit April 2021 bekannt, derzeit aber noch nicht behoben. Ein ansonsten arbeitsfähiges Python-Skript in enger Anlehnung an das Beispiel von van Strien (2015) ist in Anlage 2 ab Seite xii im Anhang zu finden. Es wurde das Python-Modul *pythonOCC* genutzt, um *IfcOpenShell* direkt mit Python nutzen zu können (Paviot, 2021). Es wurde eine Aktualisierung auf die neue API[21] von *pythonOCC* Version > 7.4 sowie *Python* Version > 3.6 vorgenommen.

---

[19] Weitere Informationen zu *OpenCascade* unter dev.opencascade.org online abrufbar (Zuletzt abgerufen am 20.08.2021)

[20] Bugreport ist als GitHub-Issue #1439 online abrufbar unter github.com/IfcOpenShell/IfcOpenShell/issues/1439 (Zuletzt abgerufen am 01.09.2021)

[21] Siehe auch github.com/tpaviot/pythonocc-core/issues/565 (Zuletzt abgerufen am 17.08.2021)

Um dennoch eine nutzbaren Karte aus dem horizontalen Schnitt zu erhalten, wurde aus dem dreidimensionalen Gebäudemodell der Simulationsumgebung, wie in Abschnitt 4.5.1 beschrieben, eine Bilddatei erzeugt. Dazu wurde die Ansicht des Schnittes von oben als Bild exportiert.

4.4.2. KONVERTIERUNG IN ROS-FÄHIGES KARTENFORMAT

Die Bilddatei des Grundrisses muss anschließend für die Navigationseinheit des Roboters angepasst werden. Das hier genutzte Modul *nav2* liest Bilddateien im Format PGM als Kartenmaterial ein (siehe auch Abschnitt 3.4.1). Da bei größeren Gebäudegrundrissen sehr große Bilddateien entstehen können, werden diese vor der Konvertierung in das PGM-Format maßstabsgerecht skaliert und der Skalierungsgrad dokumentiert. Bei der Erstellung der Bilddatei direkt mit *IfcOpenShell* ist der Maßstab bekannt, was eine automatische Skalierung möglich macht. Da jedoch auf den händischen Export des horizontalen Schnittes zurückgegriffen werden musste, ist ebenfalls eine händische Skalierung nötig. Dazu wurde das von Sears-Collins und Sperbeck (2021) vorgeschlagene Verfahren genutzt, Referenzpunkte im Bild zu markieren und deren reale Länge anzugeben. Aus diesen Angaben lassen sich der horizontale und vertikale Maßstab der Bilddatei ermitteln und entsprechend skalieren. Zum Schluss wird das so erstellte Bild in das PGM-Format konvertiert und die Metainformationen im YAML-Format erzeugt. Die Metainformationen enthalten unter anderem den zuvor ermittelten Maßstab, in der die Karte vorliegt.

Die von Sears-Collins und Sperbeck (2021) bereitgestellten Python-Skripte wurden so angepasst, dass eine beliebige Bilddatei eingelesen werden kann. Diese wird in ein binäres, schwarz-weißes Bild konvertiert, in welchem dann jeweils zwei Punkte für die Ausdehnung in x-Richtung beziehungsweise y-Richtung angeklickt werden können. Durch Angabe der realen Abstände der Punkte wird der aktuelle Maßstab der Breite (x-Richtung) und Höhe (y-Richtung) ermittelt. Anschließend wird das Bild auf zehn Prozent seiner ursprünglichen Größe verkleinert und der Maßstab entsprechend angepasst. Ein Aufruf des Skripts erfolgt, vorausgesetzt Skript und Bild befinden sich im selben Ordner und das Terminal wurde aus diesem heraus gestartet, folgendermaßen:

```
python make_ROS_map.py <building.png> -o <map name>
```

Die Angabe des Ausgabenamens der Karte ist dabei optional. Eine Installation des freien Open Source Bildkonvertierungsprograms *ImageMagick* ist ebenfalls vorausgesetzt (ImageMagick Studio LLC, 2021). Das Skript erzeugt einen neuen Ordner mit dem Präfix map_ und dem aktuellen Zeitstempel, in dem die YAML-Datei, die Kartendatei als PGM und das skalierte Bild gespeichert werden. Der Quellcode des Skripts ist im Anhang in Anlage 3 auf Seite xiv zu finden.

4.5. ERZEUGUNG DER SIMULATIONSUMGEBUNG

Um in ROS 2 eine Umgebung simulieren zu können, steht das Paket *Gazebo* zur Verfügung (Open Robotics, 2020a). Mithilfe dieser Software lassen sich eine Vielzahl von Eigenschaften, physikalischen Gesetzen und Geometrien simulieren. Dazu zählen unter anderem Lichtquellen, Masseträgheit, Reibung, Kollisionserkennung, ganze Roboter und Schwerkraft. Auch Sensoren wie beispielsweise LIDAR oder Kameras sind in der erzeugten Umgebung virtuell benutzbar. Dadurch lassen sich realitätsnahe Simulationsergebnisse für die Wegfindung, Orientierung oder allgemeine Bewegung eines Roboters im Raum erzielen. Die Realitätstreue ist dabei stark abhängig von der gewünschten Genauigkeit sowie der verwendeten Modellierung und Optimierung. So bietet ROS beispielsweise standardmäßig voroptimierte und mit realitätsnaher, zufälliger Streuung modellierte LIDAR-Sensoren an, die den Eigenschaften realer Sensoren sehr nahe kommen und als Pakete aus dem ROS-Repository zur sofortigen Nutzung herunterladbar sind.

Eine Simulationsumgebung wird in Gazebo als *World* bezeichnet. Diese kann von einer einfachen, ebenen Plattform ohne jegliche physikalische Gesetzmäßigkeit bis hin zu komplexen Modellen mit Robotern und detaillierten geometrischen Umgebungsmodellen mit Kollisionserkennung sowie Wind, Reibung und Schwerkraft reichen. In der hier genutzten Implementation wird aus der IFC-Datei ein dreidimensionales Modell aus Polygonen (Mesh) für die Simulation von Geometrie und Kollisionserkennung der Umgebung genutzt. In den folgenden Abschnitten wird erläutert, wie die hier verwendete Simulationsumgebung aus der IFC-Datei erzeugt und um Eigenschaften wie Schwerkraft und Kollisionserkennung erweitert wurde.

4.5.1. GEOMETRIE DES GEBÄUDES

Wie in Abbildung 20 im Abschnitt 4.4 bereits erwähnt, muss für die Simulation dreidimensionaler Objekte wie beispielsweise Wände, Bäume oder ganze Gebäude ein entsprechendes dreidimensionales Modell vorliegen. Da Gazebo keinen direkten Import der implizit vorliegenden IFC-Geometriedaten unterstützt, wird als Austauschformat das offene, auf dem XML-Standard basierende *COLLADA*-Format (kurz für *Collaborative Design Activity*) genutzt. Das von der Firma *Kronos Group* entwickelte Format zum Austausch von explizit definierten, dreidimensionalen Modellgeometrien kann durch das quelloffene Softwarepaket *OpenCOLLADA* verarbeitet werden (Khronos Group, 2018). Dieses kann sowohl für die optische Repräsentation innerhalb von Gazebo als auch für die Simulation physikalischer Eigenschaften des Gebäudes wie beispielsweise Reibung am Boden, Reflektion von Laserlicht eines LIDAR-Sensors oder Kollision mit dem Roboter genutzt werden.

Konvertierung mit IfcConvert
Nachfolgend ist ein Beispiel für den Aufruf von *IfcConvert* über das Terminal dargestellt,

wie er auch für die Erstellung der .dae-Dateien für die Simulation verwendet wurde. Der Befehl muss in einer einzelnen Terminalzeile ohne Zeilenumbrüche ausgeführt werden.

```
./IfcConvert --include+=arg Name "01_OG_OKFB_3,35"
--exclude=entities IfcOpeningElement IfcSpace IfcDoor
--use-element-names --use-material-names
--use-element-types --use-element-hierarchy <input-file.ifc> <output-file.dae>
```

Die Konvertierung erfolgt dabei mit der quelloffenen Software *IfcConvert*, die direkt als ausführbare Datei ohne weitere Installation nach dem Download über das Terminal nutzbar ist (Krijnen, 2021a). Die IFC-Datei kann dabei nach verschiedenen Kriterien gefiltert werden:

- Um nur Entitäten beziehungsweise physische Elemente einer bestimmten Etage in das Geometriemodell zu überführen, wird über die Filteroption `--include+=arg` in Kombination mit dem Argument `Name "<Name der Etage>"` die gewünschte Etage spezifiziert. Die Option `include+` meint dabei, dass auch alle im gefilterten Element enthaltenen Teilelemente sowie von diesem umschlossene Elemente mit beachtet werden. Dies ist notwendig, da nur das Element *IfcBuildingStorey* den Namen der Etage enthält, aber alle in der Etage befindlichen Elemente gesucht sind. Die in IFC dafür notwendigen Relationen sind *IsDecomposedBy*, *HasOpenings*, *FillsVoid* und *ContainedInStructure*.
- Die Option `--exclude=entities` ermöglicht es, bestimmte Entitäten aus der erzeugten Geometrie komplett auszuschließen. Standardmäßig werden *IfcOpeningElement* und *IfcSpace* ausgeschlossen, dadurch bleiben allerdings Türen noch im Modell enthalten. Diese werden als *IfcDoor* ebenfalls entfernt, damit diese nicht aufwendig als bewegliche Teile im 3D-Modell nachbearbeitet werden müssen.
- Die Optionen `--use-element-names --use-material-names --use-element-types --use-element-hierarchy` betreffen lediglich die interne Benennung und Sortierung der Geometrien. Sie sind für die Simulation selbst nicht notwendig. Diese Optionen führen dazu, dass anstatt zufällig generierter Geometrienamen die Material-, Typen- und Element-Namen aus der IFC-Datei verwendet werden und erleichtern das Wiederfinden und Kontrollieren von Elementen in der generierten .dae-Datei.

Nach der Konvertierung in das explizite Geometrie-Austauschformat COALLADA wurde noch ein horizontaler Schnitt durch die Etage geführt und nicht benötigte Geometrien wie Rohrleitungen oder Außenfassadenelemente entfernt. Das dient lediglich der besseren Sichtbarkeit des Roboters in der Visualisierung und verringert außerdem den nötigen Rechenaufwand während der Simulation in *Gazebo*. Diese Nachbearbeitung es Modells wurde in der Software *Blender* vorgenommen (Blender Institute, 2021). Die Höhe der Schnittführung wurde so gewählt, dass alle Türstürze nicht mehr im Modell enthalten sind.

Konvertierung mit IfcOpenShell
Alternativ zu der zuvor beschriebenen Methode ist auch hier eine Nutzung des Softwarepakets *IfcOpenShell* möglich. Wie in Abschnitt 4.4.1 bereits erläutert, verhindert jedoch auch hier der derzeit vorhandene Fehler in *IfcOpenShell* beziehungsweise der davon genutzten Bibliothek *OpenCASCADE* eine praktische Umsetzung. Die anschließende Konvertierung der so erzeugten IFC-Datei wird anschließend ebenfalls mit *IfcConvert* vorgenommen, allerdings ohne zusätzliche Filter.

Revit-Plugin
Zusätzlich besteht die Möglichkeit, das Plugin *Rhino.Inside®.Revit*[22] zu nutzen, welches das Exportieren von in *Autodesk Revit* geöffneten IFC-Modellen als Mesh erlaubt und somit auch sämtliche Filter- und Bearbeitungsoptionen von Revit für die Anpassung der Simulationsumgebung nutzbar macht. Dieser Ansatz funktionierte zwar ebenfalls, allerdings ist das Öffnen von nicht durch *Revit* erstellten IFC-Dateien in *Revit* fehlerhaft, wodurch auch der Export als .dae-Datei unvollständig sein kann. So werden beispielsweise Säulen beziehungsweise die Entitäten *IfcColumn* nicht immer übernommen und fehlen dann im Modell. Außerdem sind die genannten Programme nicht als Open Source Software verfügbar.

Mobiliar im Gebäudemodell
Das Mobiliar im Gebäudemodell des fiktiven Bürogebäudes wurde im dreidimensionalen Modell belassen. Dadurch lässt sich prüfen, wie das Navigationssystem des Roboters auf Hindernisse reagiert, die nicht in der Karte verzeichnet sind. In Abbildung 21 ist in einem Screenshot des im FZKViewer visualisierten IFC-Modells zu sehen, wie Tische und Stühle

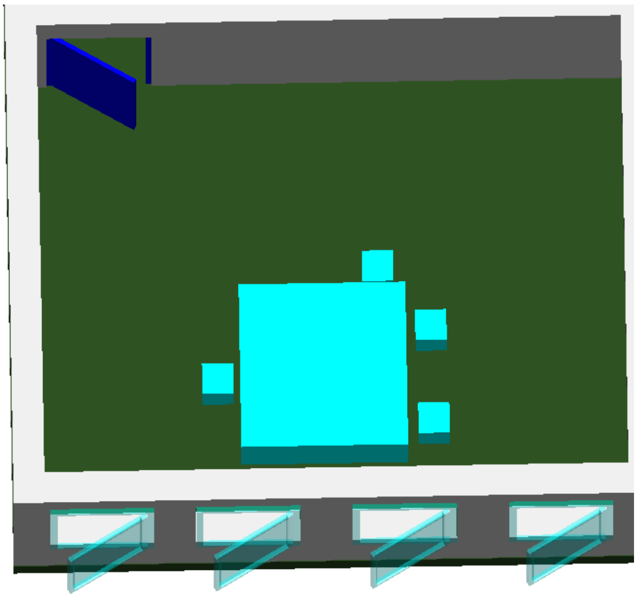


Abbildung 21 - Detailansicht eines Raumes mit Mobiliar (Screenshot aus FZK-Viewer)

---

[22] Zu finden unter rhino3d.com/inside/revit/beta/ (Zuletzt abgerufen am 20.08.2021)

als abstrakte Quader im Modell eingebunden sind. Die hellblauen Quader sind nicht in der Karte, dafür aber im Simulationsmodell enthalten.

4.5.2. KONFIGURATION DER SIMULATIONSUMGEBUNG FÜR GAZEBO

Die maschinenlesbare Beschreibung und Konfiguration einer *World* in *Gazebo* wird über eine Textdatei im XML-Datenschema realisiert. Es wird dabei das Format SDF (Simulation Description Format) verwendet, welches als Open Source Datenschema dokumentiert ist (Open Robotics, 2020d). Die Dateiendung ist beliebig wählbar, üblich sind jedoch .sdf, .world oder .model, je nach Autor des Paketes. Als Grundlage für die hier genutzte SDF-Datei diente das Paket *nav2_bringup* für ROS 2 (Macenski et al., 2021). Die Datei `/worlds/waffle.model` wurde so modifiziert, dass die Geometrie des Gebäudes sowohl visuell als auch als Kollisionsmodell in *Gazebo* beim Start der Simulation geladen wird. Auch die Positionierung des Roboters zu Simulationsbeginn wurde angepasst. In den Zeilen 46 bis 51 wurde dazu das Geometriemodell der Umgebung durch das Gebäudemodell ersetzt, welches im Ordner `/models/building_mesh/` abgelegt ist:

```
46    <model name="building_mesh">
47      <static>1</static>
48      <include>
49        <uri>model://building_mesh</uri>
50      </include>
51    </model
```

Dabei wird das in Zeile 49 definierte Modell namens *building_mesh* aus dem Modell-Ordner `/models` aufgerufen. Dieses ist durch die Datei `model.sdf` folgendermaßen definiert:

```
 1  <sdf version='1.5'>
 2    <model name="building_mesh">
 3      <pose>-3.0 -8.4 -3.0  0 0 0</pose>
 4      <static>true</static>
 5      <link name="body">
 6        <visual name="visual">
 7          <geometry>
 8            <mesh>
 9              <uri>model://building_mesh/meshes/building.dae</uri>
10            </mesh>
11          </geometry>
12        </visual>
13        <collision name="collision">
14          <geometry>
15            <mesh>
16              <uri>model://building_mesh/meshes/building.dae</uri>
17            </mesh>
18          </geometry>
19          <surface>
20            <bounce />
21            <friction>
22              <ode />
23            </friction>
24            <contact>
25              <ode />
26            </contact>
27          </surface>
28        </collision>
29      </link>
30    </model>
31  </sdf>
```

Über den Befehl `<pose>` wird der Ort und die Rotation des Gebäudemodells im Koordinatensystem von Gazebo händisch festgelegt. Dieser ist abhängig vom Koordinatenursprung des genutzten Gebäudemodells. Im hier gezeigten Beispiel wurde die Höhe der Bodenplatte des Gebäudes auf die Höhe der x-y-Ebene gelegt und der Rand des Gebäudes in den Koordinatenursprung verschoben, was aber keine Notwendigkeit darstellt. Da der Roboter im Koordinatenursprung eingesetzt wird, bietet sich dies aber an. Die Zeilen 5 bis 12 weisen dem Modell das zuvor erstellte Mesh als visuelle Repräsentation zu, die Zeilen 13 bis 18 dasselbe Mesh als physikalisches Modell zur Kollisionsermittlung.

Weiterhin ist für den Reibungskoeffizienten der Räder des Roboters ein sehr hoher Wert von $\mu = \ 10000$ angesetzt, um ein Durchdrehen dieser auszuschließen. Die Schwerkraft ist mit $g = 9.81\ {}^{m}/_{s^2}$ angesetzt.

4.6. SIMULATION IN ROS 2

Die in den vorherigen Abschnitten beschriebenen, selbst erstellten Komponenten müssen anschließend in einem *Workspace* beziehungsweise Arbeitsordner gespeichert und als Paket in ROS zusammengeführt werden.

4.6.1. PAKET FÜR ROS ERSTELLEN

Als Paketgrundlage wird wieder das Paket *nav2_bringup* genutzt, welches mit den eigenen Komponenten ergänzt beziehungsweise ersetzt wird. Das erstmalige Anlegen eines solchen Workspaces aus dem Terminal heraus wird folgendermaßen erreicht:

```
# Einrichten der Umgebungsvariablen für ROS
source /opt/ros/foxy/setup.bash

# Anlegen des Arbeitsordners
mkdir -p ~/ros2ws/da_ws/src
cd ~/ros2ws/da_ws/src

# Herunterladen…
git clone https://github.com/ros-planning/navigation2.git --branch foxy-devel
cd ~/ros2ws/da_ws
rosdep install -y -r -q --from-paths src --ignore-src --rosdistro foxy

# … und Building des Quellordners von nav2
colcon build --symlink-install
. install/setup.bash
```

In jedem ROS-Paket befindet sich ein Ordner `src/`, in welchem der Quellcode des Paketes abgelegt ist. In diesen Ordner werden die Anpassungen eingearbeitet. Im Ordner `maps/` wird die Karte, bestehend aus der zweidimensionalen Karte im PGM-Format sowie den Metainformationen im YAML-Format, unter dem Namen `turtlebot3_world` abgelegt. Im Ordner `models/building_mesh/` wird das Gebäudemodell abgelegt. Um die getätigten Aktualisierungen in das Paket einzuarbeiten, muss der veränderte Paketteil *nav2_bringup* neu erstellt werden, was als *building* bezeichnet wird:

```
source /opt/ros/foxy/setup.bash
cd ~/ros2ws/da_ws
. install/setup.bash

colcon build --packages-select nav2_bringup
```

Das so erzeugte, aktualisierte ROS-Paket kann nun in der Simulation verwendet werden.

Die Ausführung der Simulation selbst wird über einen sogenannten *Launchfile* realisiert, der alle nötigen Programmteile automatisch startet. Dazu gehören die Visualisierungsumgebung der Navigationseinheit durch das Programm *rviz*, die Simulationsumgebung in *Gazebo* und die Pakete des Roboters sowie des Navigationsmoduls *nav2*. Es wurde dazu der Launchfile `tb3_simulation_launch.py` des Paketes *nav2_bringup* verwendet, welcher sich im Ordner `bringup/` befindet. In einem neuen Terminal werden folgende Befehle ausgeführt:

```
source /opt/ros/foxy/setup.bash
cd ~/ros2ws/da_ws
. install/setup.bash

export TURTLEBOT3_MODEL=waffle
export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:<workspace path>/install/nav2_bringup/
  share/nav2_bringup/models
```

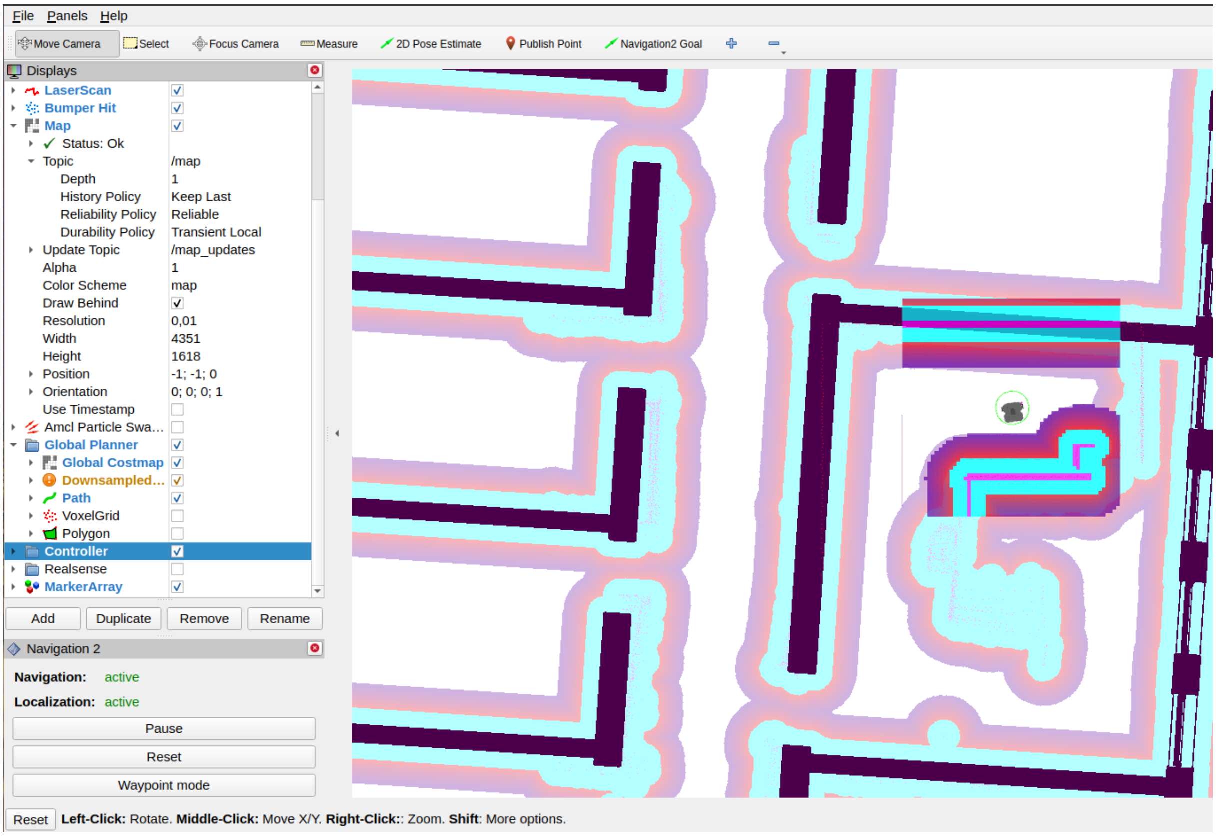
Dies initialisiert alle Umgebungsvariablen von ROS und legt den Pfad fest, in dem die zuvor veränderten und durch *building* erzeugten Paketdaten abgelegt wurden. Der Start durch den Launchfile erfolgt dann im selben Terminal durch den Befehl

```
ros2 launch nav2_bringup tb3_simulation_launch.py
```

4.6.2. NAVIGATION MIT NAV2

Nach erfolgreichem Simulationsstart muss dem Roboter in *rviz* einmalig seine aktuelle Position über die Funktion *2D Pose Estimate* in rviz mitgeteilt werden. Anschließend können mit *Navigation2 Goal* Punkte auf der Karte ausgewählt werden, die der Roboter ansteuern soll. Er sendet dazu dauerhaft die Sensordaten des LIDAR-Systems an die Navigationseinheit *nav2*, die die Daten mit der Karte abgleicht und daraus die aktuelle Position auf der Karte ermittelt. Der Weg zum Zielpunkt wird durch den A*-Algorithmus aus einer Rasterung der Karte ermittelt und aktualisiert, sobald der Roboter eine neue Position auf der Karte eingenommen hat. Der Weg über die Rasterung wird allerdings nicht direkt als Pfad genutzt, sondern algorithmisch vorher geglättet, um ruckartige Richtungswechsel zu vermeiden und die Fahrtdauer nicht zu verlängern (Macenski, Martín et al., 2020, S. 4).

Falls der Roboter keinen Abgleich zwischen Karte und Umgebung mehr ausführen kann, führt er mehrere Manöver zur Rekalibrierung durch. Diese Manöver werden als *Recovery* bezeichnet. Zuerst löscht er die lokale Costmap und scannt die Umgebung erneut. Führt das nicht zur Erkennung einer Position auf der Karte, dreht sich der gesamte Roboter langsam auf der Stelle, um andere Ausrichtungen im Verhältnis zur Karte zu testen. Führt auch das nicht zum Erfolg, sendet der Roboter ein optisches oder akustisches Signal und verharrt in seiner aktuellen Position, um auf externe Hilfe zu warten. Eine solche Situation kann beispielsweise entstehen, wenn zu viele nicht in der Karte enthaltene Hindernisse in der Umgebung des Roboters abgetastet werden. Dabei kann es sich zum Beispiel um abgestelltes Baumaterial oder eine abweichend zum BIM-Modell ausgeführte Wandanordnung handeln. Solange das LIDAR-System aber ausreichend statische, mit der Karte übereinstimmende Hindernisse erreichen kann, ist die Lokalisierung in der prinzipiell Karte möglich (Macenski, Martín et al., 2020, S. 4–6).



LaserScan	☑
Bumper Hit	☑
Map	☑
Status: Ok	
Topic	/map
Depth	1
History Policy	Keep Last
Reliability Policy	Reliable
Durability Policy	Transient Local
Update Topic	/map_updates
Alpha	1
Color Scheme	map
Draw Behind	☑
Resolution	0,01
Width	4351
Height	1618
Position	-1; -1; 0
Orientation	0; 0; 0; 1
Use Timestamp	☐
Amcl Particle Swa…	☐
Global Planner	☑
Global Costmap	☑
Downsampled…	☑
Path	☑
VoxelGrid	☐
Polygon	☐
Controller	☑
Realsense	☐
MarkerArray	☑

Navigation: active

Localization: active

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift**: More options.

Abbildung 22 - Visualisierung der Sensordaten im Programm rviz (Screenshot)

Eine wichtige Einschränkung muss für die Art des Roboterantriebes bezüglich der Wegermittlung gemacht werden. Die von nav2 ermittelten Wege setzen voraus, dass der Roboter sich auf der Stelle drehen kann. Dies lässt sich beispielsweise durch einen Kettenantrieb oder einzeln steuerbare, unabhängige Antriebsräder erreichen.

4.7. SIMULATIONSERGEBNISSE

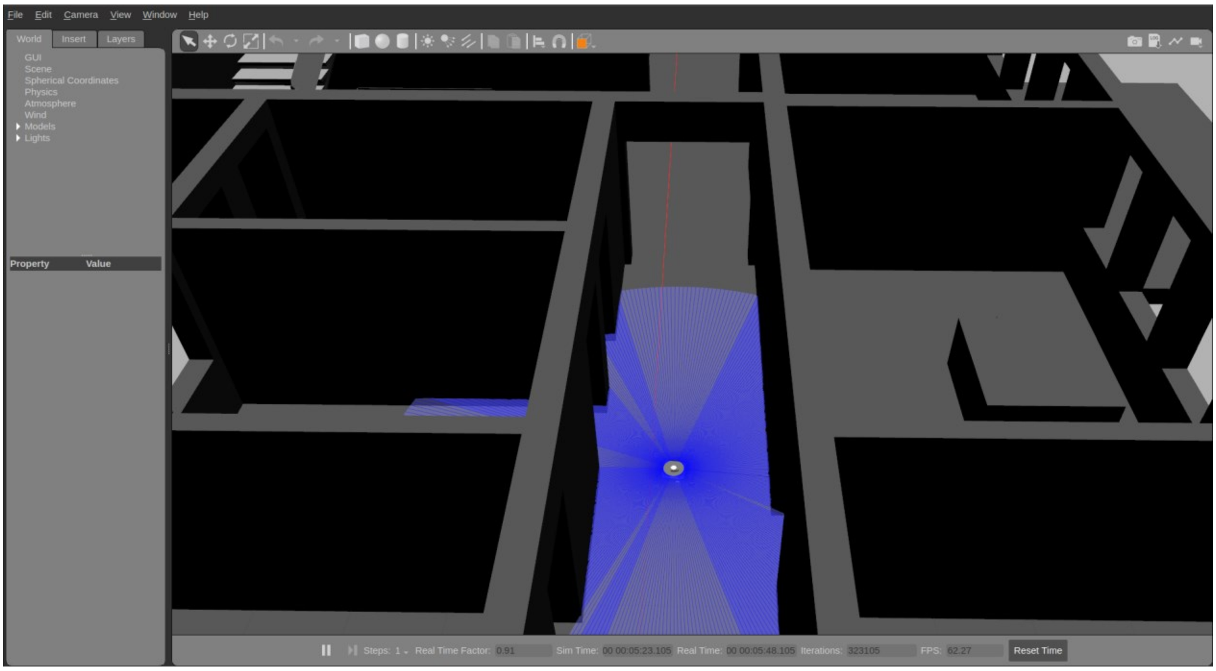
Nach den in den Abschnitten 4.4 bis 4.6 erläuterten Vorbereitungen der Simulation kann diese wie in Abschnitt 4.6.1 aus dem Terminal heraus gestartet werden. Je nach Größe der Umgebungskarte und des dreidimensionalen Umgebungsmodells kann der erstmalige Start der ROS-Hilfsprogramme *rviz* und *Gazebo* einige Sekunden bis zur vollständigen Nutzbarkeit dauern. Die Eingabe der Navigationsziele erfolgt händisch mit der Maus. Im Ausgabefenster von *Gazebo* lässt sich die Simulation visuell Nachverfolgen und die Kameraperspektive beliebig verändern. In *rviz* sind dagegen die Sensor- und Kartendaten des Roboters visualisiert.

Der Screenshot in Abbildung 22 zeigt das Programm *rviz* während einer laufenden Simulation. Angezeigt werden:

- Der Roboter selbst als graues Modell sowie dessen Mindestscanabstand der Sensoren als grüner Ring
- Die Karte des Gebäudes als schwarzer-weißer Hintergrund
- Durch das LIDAR-System ermittelte Abstände der Umgebung des Roboters als rote Punkte
- Anhand der Kartendaten sowie aus lokalen Scans durch den globalen Planungsalgorithmus (*Global Planner*) im Voraus geplante Sicherheitsabstände von Hindernissen als hellblaue und rote, leicht transparente Flächen
- Anhand der lokalen Scanergebnisse des LIDAR-Systems ermittelte lokale Hindernisse (rosa) sowie Sicherheitsabstände (hellblau und rot)

Zusätzlich lassen sich noch einzelne Partikel des AMCL-Paketes, das Blickfeld und aktuelle Bild der optischen Kamera sowie der derzeit vom Roboter geplante Weg anzeigen.

In Gazebo ist neben dem Roboter und dem Gebäudemodell zusätzlich eine Visualisierung des LIDAR-Systems dargestellt. So lässt sich an den hellblauen Linien erkennen, welcher Scanbereich derzeit abgedeckt ist und an den hellblauen Linien, welche Laserstrahlen reflektiert und detektiert wurden. In Abbildung 23 ist an den hellen Linien links gut zu erkennen, an welcher Stelle sich Türen im Flur befinden, da in dem Modell der Simulationsumgebung die Türöffnungen leer sind und sich dementsprechend keine Türen als Hindernis darin befinden. Der Roboter selbst ist als grauer Zylinder im Zentrum der LIDAR-Strahlen dargestellt.



Steps: 1 Real Time Factor: 0.91 Sim Time: 00 00:05:23.105 Real Time: 00 00:05:48.105 Iterations: 323105 FPS: 62.27

Abbildung 23 - Visualisierung der Simulation in Gazebo (Screenshot)

4.7.1. GENAUIGKEIT DER LOKALISIERUNG

Grundsätzlich wird in der Simulation für die globale Lokalisierung des Roboters in der Karte, das sogenannte *map-matching*, ein AMCL-Algorithmus genutzt. Für lokale, also in Scanreichweite des Roboters liegende und unkartierte Hindernisse wird zusätzlich ein lokaler SLAM-Algorithmus angewendet, um diese in die globale Karte einzubinden. Ist ein so erkanntes Hindernis durch Fortbewegung des Roboters oder Verdeckung durch ein neues Hindernis nicht mehr in Scanreichweite des Roboters, wird nach einer gewissen Zeit diese Information wieder aus der Karte entfernt. Dadurch wird sichergestellt, dass auch eine spätere Entfernung des Hindernisses wieder in die Karte einfließen kann. Würden durch den SLAM-Algorithmus gefundene Hindernisse dauerhaft als nicht befahrbares Gebiet vermerkt werden, würden temporäre Hindernisse immer wieder in der globalen Routenplanung berücksichtigt werden, obwohl sie eventuell gar nicht mehr den Weg blockieren. Die Größe der aus dem SLAM-Algorithmus ermittelten, lokalen *Costmap* um den Roboter herum ist für den *TurtleBot3 Waffle Pi* standardmäßig ein 3 mal 3 Meter großes Quadrat mit einer Auflösung von 5 Zentimetern pro Pixel, kann aber in der Konfigurationsdatei des *nav2*-Paketes `params/nav2_params.yaml` im Abschnitt `local_costmap` verändert werden. Für den Roboter selbst wird ein sogenannter *footprint* im Mittelpunkt der lokalen Costmap angenommen. Dies ist die virtuelle Grundfläche des Roboters, die er im Einsatz einnimmt. Für den *TurtleBot3 Waffle Pi* ist sie als Kreis mit einem Radius von 22 Zentimetern angesetzt. Die Grundfläche kann aber prinzipiell auch genauer modelliert werden, falls der eingesetzte Roboter eine andere Form besitzt.

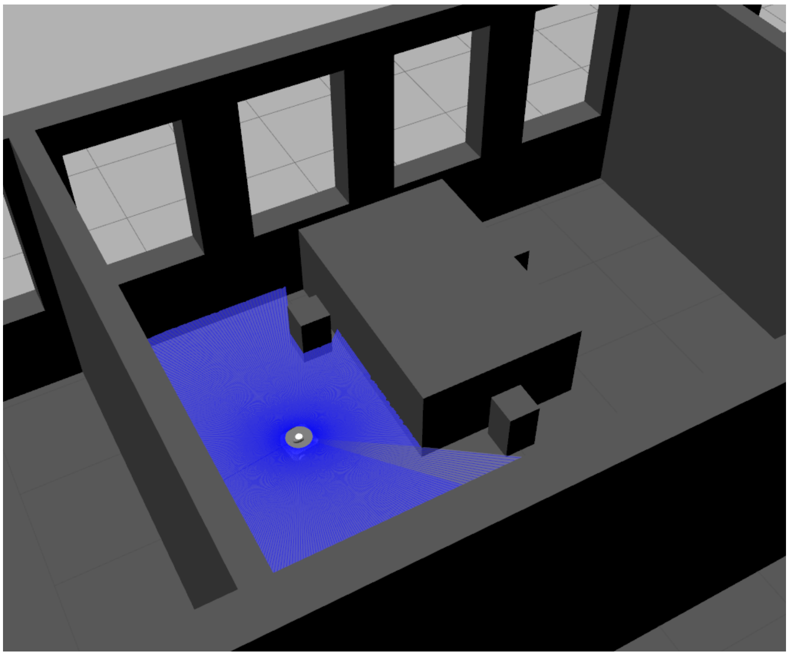


Abbildung 24 - Hindernisse in einem Raum (Screenshot Gazebo)

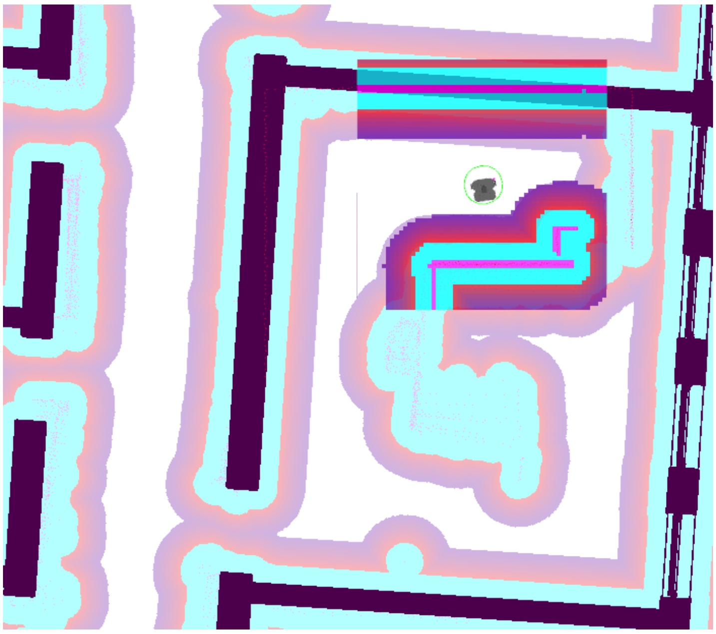


Abbildung 25 - Globales map-matching und lokales SLAM in rviz (Screenshot)

In Abbildung 25 ist der lokale SLAM-Bereich an der stärkeren Farbgebung erkennbar, während der globale Bereich für globale Routenplanung und map-matching transparente Farben aufweist. Das im Vorbeifahren lokal erkannte Hindernis durch Tische und Stühle (siehe auch die Visualisierung derselben Situation in Abbildung 24) wurde temporär in die Karte eingetragen, ein Teil davon ist auch noch im Scanbereich des Roboters sichtbar. Dabei ist in Abbildung 25 am rechten Bildrand gut erkennbar, dass in der Karte die vom Roboter erkannte Wand eigentlich weiter rechts sein müsste. Diese Abweichung ergibt sich vor allem durch die alleinige Nutzung des AMCL-Algorithmus zur globalen Lokalisierung. Die Räume bieten häufig nicht genug eindeutige geometrische Merkmale für den Tiefenscan, sodass es oft mehrere wahrscheinliche Positionen auf der Karte gibt, in denen sich der Roboter befinden könnte. Dadurch „springt“ die ermittelte Position manchmal um mehrere Zentimeter oder sogar Meter. Dieses Verhalten lässt sich durch eine höhere Reichweite des LIDAR-Systems verringern, da so mehr beziehungsweise weiter entfernte geometrische Eigenschaften der Umgebung umfasst werden können. Bei größeren Räumen mit weit auseinanderliegenden Wänden versagt die Lokalisierung völlig, da keine Referenzpunkte mehr vorhanden sind, an denen sich orientiert werden kann. Die Reichweite des LIDAR-Systems sollte im Idealfall größer sein, als der größtmögliche Abstand zu umgebenden Wänden im Gebäude.

Doch auch mit größeren Scanreichweiten des LIDAR erwies sich die alleinige Lokalisierung mithilfe des AMCL-Algorithmus als relativ ungenau. Abweichungen von mehreren Zentimetern bis Dezimetern von der realen Position waren fast immer vorhanden. Die üblicherweise rechteckigen Räume eines Gebäudes bieten zu wenig einzigartige Referenzpunkte für den AMCL-Algorithmus. Unkartierte Hindernisse verringern die Genauigkeit zusätzlich. Weiterhin sind eine Vielzahl Parameter[23] für das *nav2*-Paket und dessen Add-Ons anpassbar und müssen für jeden Anwendungsfall optimiert werden. Für kleinere Räume mit bereits kartierten Referenzpunkten innerhalb des Raumes wie beispielsweise Mobiliar, Säulen oder mehreren Türöffnungen kann eine zentimetergenaue Lokalisierung erreicht werden, darüber hinaus jedoch nicht. Zum Zeitpunkt des Innenausbaus eines Gebäudes kommen dazu lediglich Säulen und (Tür-) Öffnungen in Höhe des LIDAR-Systems in Frage, welche in der Praxis jedoch selten in ausreichendem Maße vorhanden sind.

4.7.2. ROBUSTHEIT DER NAVIGATION

Für wenige Meter oder wenige Räume entfernte Zielorte war die Ermittlung des optimalen Weges fast in Echtzeit verfügbar und führte oft zur erfolgreichen Fahrt des Roboters an die gewünschte Stelle mit der gewünschten Endposition. In Abbildung 26 ist beispielsweise der geplante Weg vom Eingang des Raumes in den Bereich zwischen oberer Wand und Tisch als rötliche Linie visualisiert. Die Wegplanung ist so eingerichtet, dass Hindernisse möglichst außerhalb des Sicherheitsbereiches umfahren werden. Es ist jedoch auch deutlich zu sehen, dass die Abweichungen des map-matching zur realen Position des

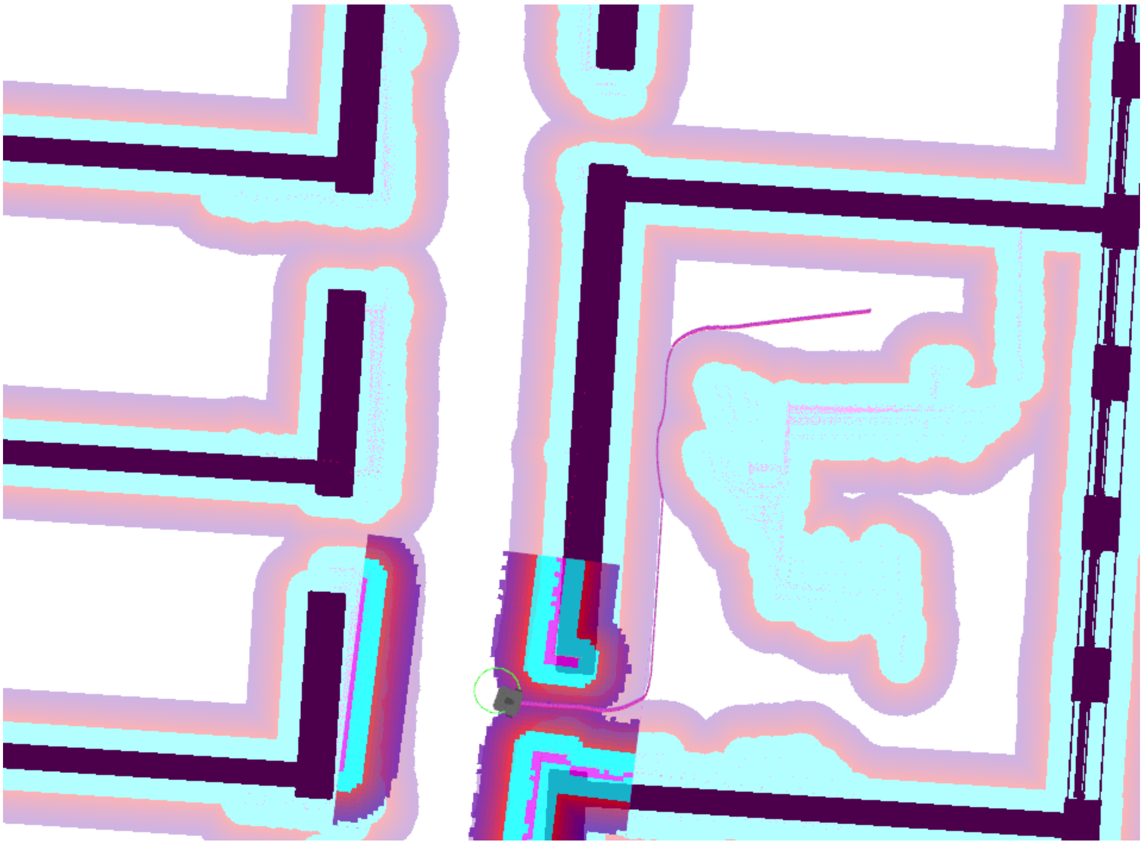


Abbildung 26 - Navigationspfad entlang von Hindernissen (Screenshot rviz)

---

[23] Für weitere Details siehe auch navigation.ros.org/configuration/index.html (Zuletzt abgerufen am 19.08.2021)

Roboters immer größer werden. In Abbildung 26 ist dies an den immer weiter im Uhrzeigersinn abweichenden roten Punkten des LIDAR-Scans erkennbar. Dies führte vor allem bei längeren Geradeausfahrten zu „Zick-Zack“-artigen Wegen, da die ermittelte Position des Roboters immer wieder um mehrere Zentimeter korrigiert werden musste. Weiter entfernte Zielorte konnten häufig nicht angesteuert werden, da vor allem in länglichen Fluren durch das im vorherigen Abschnitt erwähnte „Springen“ der ermittelten Position auch die Wegfindung gestört und teilweise unmöglich wurde. Die Robustheit der Navigation hängt maßgeblich von der Lokalisierungsgenauigkeit und dauerhaften Verlässlichkeit des map-matchings ab. Im Falle einer wenigen Zentimeter genauen Lokalisierung war die Navigation jedoch problemlos möglich. Auch größere, unkartierte Hindernisse stellen kein Problem dar, solange die Lokalisierung auch hier nicht maßgeblich beeinflusst wird. Stellt sich ein ermittelter Weg unerwartet als Sackgasse heraus, werden sofort alternative Wege gesucht.

5 SCHLUSSBETRACHTUNG

In diesem Kapitel werden die Erkenntnisse der vorliegenden Arbeit noch einmal zusammengefasst, ein Ausblick für mögliche weitere Optimierungen gegeben und weiterführende Untersuchungen vorgeschlagen. Weiterhin werden zusammenfassend sechs Thesen zur Diplomarbeit aufgestellt.

5.1. ZUSAMMENFASSUNG UND AUSBLICK

Die Erstellung von geometrischem Kartenmaterial aus vorhandenen BIM-Modellen ist durch die Standardisierung des Datenaustauschformates IFC auf mehrere Arten möglich. Für die Umwandlung von Dateien entsprechend dem IFC-Standard in nutzbare Grafiken stehen eine Auswahl von proprietären und quelloffenen Programmen zur Verfügung. Eine vollautomatisierte Konvertierung erfordert jedoch einen hohen Grad an Optimierung der einzelnen Umwandlungsschritte sowie umfangreiche Tests zur Plausibilität des Outputs.

Die zusätzliche Anreicherung des Kartenmaterials durch semantische Informationen bringt nur dann einen Vorteil, wenn diese für Lokalisierung und Navigation in Betracht gezogen werden können. Dies ist für eine rein geometrische Lokalisierung aus Tiefenmessungen nur sehr bedingt möglich.

Die Ergebnisse dieser Diplomarbeit haben gezeigt, dass der Einsatz von mobilen, autonomen Robotern auf Baustellen innerhalb von Gebäuden prinzipiell möglich ist. Eine alleinige Nutzung von Tiefenmessungen der Umgebungsgeometrie stellte sich allerdings nicht als ausreichend heraus. Dies gilt vor allem dann, wenn sich Raumgrundrisse innerhalb eines Gebäudes ähnlich sind, keine oder wenige signifikante Referenzpunkte vorhanden sind oder Wände weit von der aktuell ermittelten Position entfernt sind. Weitere Informationsquellen sind notwendig, um ein robustes, autonomes Robotersystem durch Gebäude navigieren zu können. Durch die dynamischen Bedingungen auf Baustellen ist ein dauerhaft zu installierendes und einzumessendes Ortungssystem nicht praktikabel. Dessen Installation und anschließender Schutz vor physikalischen, chemischen und mechanischen Einwirkungen während der Bauphase sowie dauerhafte Stromversorgung sind im praktischen Einsatz unrealistisch. Zukünftig könnten Sender im ultra-wideband-Bereich (UWB) an der Außenfassade zumindest als zusätzliche Datenübertragungs- und Ortungshilfe eine Rolle spielen, wobei aber auch hier Installation, Schutz und Stromversorgung schwierig sind.

Eine umfangreiche Optimierung der Parameter des AMCL-Algorithmus kann weiterhin die Lokalisierungsgenauigkeit erhöhen. Da die Informationen hierfür jedoch letztlich ebenfalls maßgeblich von der Verfügbarkeit von Orientierungsmöglichkeiten innerhalb der Umgebung abhängen, ist für eine genauere Lokalisierung beziehungsweise Platzierung und Ausrichtung des Roboters am gewünschten Einsatzort zusätzliche externe Hilfe notwendig. Eine Nutzung der Odometrie des Roboters ist ebenfalls denkbar, indem beispielsweise eine zuvor errechnete Strecke ausgehend von einer nahen Wand abgefahren wird. Dieses Verfahren ist jedoch nur bei eindeutigen, nahen Referenzpunkten möglich, da durch Fertigungstoleranzen, Reibung und Temperaturdehnung der Antriebe und Räder nur Schätzungen des zurückgelegten Weges möglichsind. Abweichungen von der geplanten Fahrtstrecke und gewünschtem Fahrwinkel wirken sich bei längeren Fahrten immer stärker aus. Größere Roboter oder solche, die sich auch nicht auf der Stelle drehen lassen, können auch nicht ohne weiteres nah an Wänden fahren.

Um ein wirklich in der Praxis einsetzbares Produkt anbieten zu können, muss die gesamte Informationskette automatisierbar und dabei in sich konsistent sein. Es dürfen keine relevanten Informationen verloren gehen und ein manuelles Eingreifen in den Konvertierungsprozess sollte vermieden werden. Nur dann kann ein regelmäßig aktualisiertes BIM-Modell sinnvoll für Lokalisation und Navigation von autonomen, mobilen Robotern eingesetzt werden. Dazu müssen Schnittstellen geschaffen werden, die von der IFC-Datei über die Erstellung der Karte bis hin zur Übertragung der Daten zum Roboter eine robuste Implementierung bieten.

5.2. THESEN ZUR DIPLOMARBEIT

These 1: Die Nutzung autonomer Roboter auf Baustellen des Hochbaus ist durch einzigartige, dynamische Einsatzumgebungen geprägt.

These 2: Das Fehlen von lokalen Ortungssystemen sowie fehlender Zugang zu GPS-Ortungsdiensten erfordert eine überwiegend optische Lokalisierung durch umfangreiche Scans der Arbeitsumgebung mit Entfernungsmessgeräten und Kameras.

These 3: Die umfangreiche Nutzung von BIM-Modellen zur Planung von Bauwerken bietet für die Lokalisation und Navigation von autonomen Robotern zusätzliche wertvolle Informationen, die das Fehlen herkömmlicher Ortungssysteme ausgleichen müssen.

These 4: Für einen baupraktischen Einsatz autonomer, mobiler Roboter ist derzeit noch weitreichende Forschung und ein hoher Optimierungsgrad nötig.

These 5: Der Einsatz von Robotern auf Baustellen erfordert möglicherweise ein massives Umdenken in der Planung der Bauausführung sowie der Baustelleneinrichtung.

These 6: Besonders bei großen Bauvorhaben kann der Einsatz autonomer, mobiler Roboter dem Mangel an Arbeitskräften teilweise entgegenwirken.

VI LITERATURVERZEICHNIS

Agarwal, R., Chandrasekaran, S. & Sridhar, M. (2016). *Imagining construction's digital future: The industry needs to change; here's how to manage it.* McKinsey & Company. https://www.mckinsey.com/business-functions/operations/our-insights/imagining-constructions-digital-future (Zuletzt abgerufen am 04.05.2021).

Blender Institute. (2021). *Blender* (Version 2.82.7) [Computer software]. Blender Foundation. Amsterdam, Niederlande. https://www.blender.org/

Blum, H. (1967). A Transformation for extracting new descriptors of shape, 362–380. http://pageperso.lif.univ-mrs.fr/~edouard.thiel/rech/1967-blum.pdf

Borkowski, A., Siemiątkowska, B. & Szklarski, J. (2010). Towards Semantic Navigation in Mobile Robotics. In G. Engels, C. Lewerentz, W. Schäfer, A. Schürr & B. Westfechtel (Hrsg.), *Graph Transformations and Model-Driven Engineering: Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday* (S. 719–748). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-17322-6_30

Bretto, A. (2013). *Hypergraph Theory*. Springer International Publishing. https://doi.org/10.1007/978-3-319-00080-0

buildingSMART. (2021a). *Industry Foundation Classes (IFC) - An Introduction.* buildingSMART International. https://www.buildingsmart.org/about/openbim/openbim-definition/ (Zuletzt abgerufen am 12.07.2021).

buildingSMART. (2021b). *What is openBIM?* buildingSMART International. https://www.buildingsmart.org/about/openbim/openbim-definition/ (Zuletzt abgerufen am 08.07.2021).

Carra, G., Argiolas, A., Bellissima, A., Niccolini, M. & Ragaglia, M. (2018). Robotics in the Construction Industry: State of the Art and Future Opportunities. In J. Teizer & M. König (Hrsg.), *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC), Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC).* International Association for Automation and Robotics in Construction (IAARC). https://doi.org/10.22260/ISARC2018/0121

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F. & Ranzuglia, G. (2020). *MeshLab* (Version v2020.03) [Computer software]. Visual Computing Lab of CNR-ISTI. https://www.meshlab.net/

Deng, Y., Hong, H., Deng, H. & Luo, H. (2017). BIM-Based Indoor Positioning Technology Using a Monocular Camera. In M.-Y. Cheng, H.-M. Chen & K. C. Chiu (Hrsg.), *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC), Proceedings of the 34th International Symposium on Automation and Robotics in Construction (ISARC).* International Association for Automation and Robotics in Construction (IAARC). https://doi.org/10.22260/ISARC2017/0142

Durrant-Whyte, H. & Henderson, T. C. (2016). Multisensor Data Fusion. In B. Siciliano & O. Khatib (Hrsg.), *Springer Handbook of Robotics* (S. 867–896). Springer International Publishing. https://doi.org/10.1007/978-3-319-32552-1_35

Follini, C., Magnago, V., Freitag, K., Terzer, M., Marcher, C., Riedl, M., Giusti, A. & Matt, D. T. (2021). BIM-Integrated Collaborative Robotics for Application in Building Construction and Maintenance. *Robotics*, *10*(1), 2. https://doi.org/10.3390/robotics10010002

Fox, D., Burgardy, W., Dellaert, F. & Thrun, S. (1999). Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In Association for the Advancement of Artificial Intelligence (Vorsitz), *The Sixteenth National Conference on Artificial Intelligence.* Symposium im Rahmen der Tagung von Association for the Advancement of Artificial Intelligence, Orlando, Florida, USA. https://www.cs.washington.edu/publications/monte-carlo-localization-efficient-position-estimation-mobile-robots

Ha, I., Kim, H [Hongjo], Park, S. & Kim, H [Hyoungkwan] (2018). Image-based Indoor Localization Using BIM and Features of CNN. In J. Teizer & M. König (Hrsg.), *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC), Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC).* International Association for Automation and Robotics in Construction (IAARC). https://doi.org/10.22260/ISARC2018/0107

Hamieh, A., Deneux, D. & Tahon, C. (2017). BiMov: BIM-Based Indoor Path Planning. In B. Eynard, V. Nigrelli, S. M. Oliveri, G. Peris-Fajarnes & S. Rizzuti (Hrsg.), *Lecture Notes in Mechanical Engineering. Advances on Mechanics, Design Engineering and Manufacturing* (S. 889–899). Springer International Publishing. https://doi.org/10.1007/978-3-319-45781-9_89

Haun, M. (2013). *Handbuch Robotik*. Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-39858-2

ImageMagick Studio LLC. (2021). *ImageMagick* (Version 6.9.10-23) [Computer software]. ImageMagick Studio LLC. https://imagemagick.org/script/command-line-tools.php

Ismail, A., Nahar, A. & Scherer, R. (2017). Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard. In The University of Nottingham (Vorsitz), *International Workshop on Intelligent Computing in Engineering.* Symposium im Rahmen der Tagung von The University of Nottingham, Nottingham, UK.

Ismail, A., Strug, B. & Ślusarczyk, G. (2018). Building Knowledge Extraction from BIM/IFC Data for Analysis in Graph Databases. In L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz & J. M. Zurada (Hrsg.), *Artificial Intelligence and Soft Computing: 17th International Conference, ICAISC 2018, Zakopane, Poland, June 3-7, 2018, Proceedings, Part II* (S. 652–664). Springer International Publishing.

ISO International Organization for Standardization (2016-03). *Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure* (ISO 10303-21:2016). https://www.iso.org/standard/63141.html

ISO International Organization for Standardization (2018-11). *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries - Part 1: Data schema* (ISO 16739-1:2018). https://www.iso.org/standard/70303.html

Kadena, E., Nguyen, H. & Ruiz, L. (2021). Mobile Robots: An Overview of Data and Security. In *Proceedings of the 7th International Conference on Information Systems Security and Privacy* (S. 291–299). SCITEPRESS - Science and Technology Publications. https://doi.org/10.5220/0010174602910299

Kahar, S., Sulaiman, R., Prabuwono, A. S., Amran, M. F. M. & Marjudi, S. (2011). Data transferring technique for mobile robot controller via mobile technology. In ICPAIR 2011 Organizing Committees (Hrsg.), *2011 International Conference on Pattern Analysis and Intelligence Robotics* (S. 103–108). IEEE. https://doi.org/10.1109/ICPAIR.2011.5976920

Karimi, S. & Iordanova, I. (2021). Integration of BIM and GIS for Construction Automation, a Systematic Literature Review (SLR) Combining Bibliometric and Qualitative Analysis. *Archives of Computational Methods in Engineering*, 1–22. https://doi.org/10.1007/s11831-021-09545-2

Karlsruhe Institute of Technology. (2020). *FZKViewer* (Version 6.1 Build 1816) [Computer software]. Karlsruhe Institute of Technology (KIT). Karlsruhe. https://www.iai.kit.edu/english/1648.php

Karlsruhe Institute of Technology. (2021). *KIT IFC Examples: Office Building (IFC STEP File)*. https://www.ifcwiki.org/index.php?title=KIT_IFC_Examples (Zuletzt abgerufen am 29.08.2021).

Kerbitz, C. (2019). *BIM-basierte Rettungswegermittlung mittels Graphentheorie* [Diplomarbeit]. Technische Universität Dresden, Dresden. https://tu-dresden.de/bu/bauingenieurwesen/cib/forschung/publikationen/projekt-und-diplomarbeiten

Khronos Group. (2018). *OpenCOLLADA* (Version v1.6.68) [Computer software]. GitHub, Inc. http://opencollada.com/

Konolige, K. & Nüchter, A. (2016). Range Sensing. In B. Siciliano & O. Khatib (Hrsg.), *Springer Handbook of Robotics* (S. 783–810). Springer International Publishing. https://doi.org/10.1007/978-3-319-32552-1_31

Krijnen, T. (2021a). *IfcConvert* (Version 0.6.0) [Computer software]. GitHub, Inc. http://ifcopenshell.org/ifcconvert

Krijnen, T. (2021b). *IfcOpenShell* (Version 0.6.0b0 (OCC 7.3.0)) [Computer software]. GitHub, Inc. http://ifcopenshell.org/

Lee, D.-T. (1982). Medial axis transformation of a planar shape. *IEEE transactions on pattern analysis and machine intelligence*, *4*(4), 363–369. https://doi.org/10.1109/TPAMI.1982.4767267

Lee, J. (2004). A Spatial Access-Oriented Implementation of a 3-D GIS Topological Data Model for Urban Entities. *GeoInformatica*, *8*(3), 237–264. https://doi.org/10.1023/B:GEIN.0000034820.93914.d0

Lin, F., Kerbitz, C. & Fu, Y. (2019). BIM based two-dimensional floor plan simulation and planning for evacuation. In M. Sternal, L.-C. Ungureanu, L. Böger & C. Bindal-Gutsche (Hrsg.), *31. Forum Bauinformatik: 11. bis 13. September 2019 in Berlin : proceedings* (S. 109–116). Universitätsverlag der TU Berlin. https://www.researchgate.net/publication/335977298_BIM_based_two-dimensional_floor_plan_simulation_and_planning_for_evacuation

Lin, W. Y., Lin, P. H. & Tserng, H. P. (2017). Automating the Generation of Indoor Space Topology for 3D Route Planning Using BIM and 3D-GIS Techniques. In M.-Y. Cheng, H.-

M. Chen & K. C. Chiu (Hrsg.), *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC), Proceedings of the 34th International Symposium on Automation and Robotics in Construction (ISARC).* International Association for Automation and Robotics in Construction (IAARC). https://doi.org/10.22260/ISARC2017/0060

Liu, L. & Zlatanova, S. (2011, 3. Mai). A "door-to-door" path-finding approach for indoor navigation. In International Society for Photogrammetry and Remote Sensing (Vorsitz), *GeoInformation for Disaster Management.* Symposium im Rahmen der Tagung von International Society for Photogrammetry and Remote Sensing (ISPRS), Antalya, Turkey. https://www.isprs.org/proceedings/2011/Gi4DM/PDF/OP05.pdf

Macenski, S. (2019). *On Use of SLAM Toolbox: A Fresh(er) look at Mapping and Localization for the Dynamic World*. Vortrag im Rahmen der ROSCon 2019. https://roscon.ros.org/2019/talks/roscon2019_slamtoolbox.pdf

Macenski, S., Delsey, C. & White, R. (2020). *NAV 2: Getting Started*. https://navigation.ros.org/getting_started/index.html (Zuletzt abgerufen am 15.06.2021).

Macenski, S., Jeronimo, M. & Orduno, C. (2021). *nav2_bringup* (Version 0.4.7) [Computer software]. https://index.ros.org/p/nav2_bringup/

Macenski, S., Martín, F., White, R. & Clavero, J. G. (2020, 1. März). *The Marathon 2: A Navigation System*. https://navigation.ros.org/

Nahangi, M., Heins, A., McCabe, B. & Schoellig, A. (2018). Automated Localization of UAVs in GPS-Denied Indoor Construction Environments Using Fiducial Markers. In J. Teizer & M. König (Hrsg.), *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC), Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC).* International Association for Automation and Robotics in Construction (IAARC). https://doi.org/10.22260/ISARC2018/0012

Neges, M., Wolf, M., Propach, M., Teizer, J. & Abramovici, M. (2017). Improving Indoor Location Tracking Quality for Construction and Facility Management. In M.-Y. Cheng, H.-M. Chen & K. C. Chiu (Hrsg.), *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC), Proceedings of the 34th International Symposium on Automation and Robotics in Construction (ISARC).* International Association for Automation and Robotics in Construction (IAARC). https://doi.org/10.22260/ISARC2017/0012

Nitta, Y., Yenet Bogale, D., Kuba, Y. & Tian, Z. (2020). Evaluating SLAM 2D and 3D Mappings of Indoor Structures. In K. Tateyama, K. Ishii & F. Inoue (Hrsg.), *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC), Proceedings of the 37th International Symposium on Automation and Robotics in Construction*

(ISARC). International Association for Automation and Robotics in Construction (IAARC). https://doi.org/10.22260/ISARC2020/0113

Open Cascade S.A.S. (2020). *Open CASCADE Technology (OCCT)* (Version 7.5.2) [https://dev.opencascade.org/release]. GitHub, Inc. https://dev.opencascade.org/doc/overview/html/index.html

Open Robotics. (2019, 23. Mai). *ROS 2 Design.* Open Robotics. https://design.ros2.org/ (Zuletzt abgerufen am 21.07.2021).

Open Robotics. (2020a). *Gazebo* (Version 11.5.1) [Computer software]. Open Robotics. http://gazebosim.org/

Open Robotics. (2020b). *ROS* (Version Noetic Ninjemys) [Computer software]. Open Robotics. http://wiki.ros.org/noetic/Installation

Open Robotics. (2020c). *ROS 2* (Version Foxy Fitzroy) [Computer software]. Open Robotics. https://docs.ros.org/en/foxy/Installation.html

Open Robotics. (2020d). *SDFormat: Specification (Version 1.5)*. http://sdformat.org/spec?ver=1.5 (Zuletzt abgerufen am 03.08.2021).

Open Robotics. (2021a, 15. Mai). *Distributions: List of current and historic ROS 1 distributions.* Open Robotics. https://wiki.ros.org/Distributions (Zuletzt abgerufen am 21.07.2021).

Open Robotics. (2021b, 11. Juni). *Distributions: List of current and historic ROS 2 distributions.* Open Robotics. https://docs.ros.org/en/foxy/Releases.html (Zuletzt abgerufen am 21.07.2021).

Open Robotics. (2021c, 19. Juli). *The ROS 2 graph.* Open Robotics. https://docs.ros.org/en/foxy/Tutorials/Understanding-ROS2-Nodes.html#the-ros-2-graph (Zuletzt abgerufen am 22.07.2021).

Palacz, W., Ślusarczyk, G., Strug, B. & Grabska, E. (2019). Indoor Robot Navigation Using Graph Models Based on BIM/IFC. In L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz & J. M. Zurada (Hrsg.), *Lecture Notes in Computer Science. Artificial Intelligence and Soft Computing: 18th International Conference, ICAISC 2019, Zakopane, Poland, June 16–20, 2019, Proceedings, Part II* (Bd. 11509, S. 654–665). Springer International Publishing. https://doi.org/10.1007/978-3-030-20915-5_58

Park, J., Cho, Y. K. & Martinez, D. (2016). A BIM and UWB integrated Mobile Robot Navigation System for Indoor Position Tracking Applications. *Journal of Construction Engineering and Project Management*, *6*(2), 30–39. https://doi.org/10.6106/JCEPM.2016.6.2.030

Paviot, T. (2021). *pythonOCC* (Version 7.5.1) [https://github.com/tpaviot/pythonocc-core/releases]. GitHub, Inc. https://github.com/tpaviot/pythonocc

Przybylski, M. & Siemiątkowska, B. (2012). A New CNN-Based Method of Path Planning in Dynamic Environment. In L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh & J. M. Zurada (Hrsg.), *Artificial Intelligence and Soft Computing* (S. 484–492). Springer Berlin Heidelberg.

Quigley, M., Gerkey, B. & Smart, W. D. (2015). *Programming robots with ROS* (1. Aufl.). O'Reilly & Associates Incorporated.

Ravichandran, N. (2013). *ROS - An Opensource Robotic Framework.* https://doi.org/10.13140/RG.2.2.28424.93446

Robotis, Inc. (2021a). *LDS-01: 360 Laser Distance Sensor.* Robotis, Inc. https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/ (Zuletzt abgerufen am 31.08.2021).

Robotis, Inc. (2021b). *TurtleBot3* (Version 2.1.1) [Computer software]. GitHub, Inc. https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

Scherer, R. J. & Schapke, S.-E. (2014). *Informationssysteme im Bauwesen 1*. Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-40883-0

Sears-Collins, A. L. & Sperbeck, C. (2021). *How to Create a Map for ROS From a Floor Plan or Blueprint*. https://automaticaddison.com/how-to-create-a-map-for-ros-from-a-floor-plan-or-blueprint/ (Zuletzt abgerufen am 27.08.2021).

Shan, J. & Toth, C. K. (2017). *Topographic Laser Ranging and Scanning: Principles and processing: Principles and Processing*. Second Edition. CRC Press. https://www.taylorfrancis.com/books/edit/10.1201/9781315154381/topographic-laser-ranging-scanning-jie-shan-charles-toth https://doi.org/10.1201/9781315154381

Siciliano, B. & Khatib, O. (Hrsg.). (2016). *Springer Handbook of Robotics*. Springer International Publishing. https://doi.org/10.1007/978-3-319-32552-1

Siemiątkowska, B., Harasymowicz-Boggio, B., Przybylski, M., Różańska-Walczuk, M., Wiśniowski, M. & Kowalski, M. (2013). BIM Based Indoor Navigation System of Hermes

Mobile Robot. In V. Padois, P. Bidaud & O. Khatib (Hrsg.), *CISM International Centre for Mechanical Sciences, Romansy 19 - Robot Design, Dynamics and Control: Proceedings of the 19th CISM-Iftomm Symposium* (S. 375–382). Springer Vienna.

Stachniss, C., Leonard, J. J. & Thrun, S. (2016). Simultaneous Localization and Mapping. In B. Siciliano & O. Khatib (Hrsg.), *Springer Handbook of Robotics* (S. 1153–1175). Springer International Publishing. https://doi.org/10.1007/978-3-319-32552-1_46

Steger, A. (2007). *Kombinatorik - Graphentheorie - Algebra* (2. Aufl.). *Diskrete Strukturen: Bd. 1*. Springer-Verlag Berlin Heidelberg.

Syta, H. & van de Weygaert, R. (2009, 16. Dezember). *Life and Times of Georgy Voronoi*. http://arxiv.org/pdf/0912.3269v1

Taneja, S., Akinci, B., Garrett, J. H. & Soibelman, L. (2016). Algorithms for automated generation of navigation models from building information models to support indoor map-matching. *Automation in Construction*, *61*, 24–41. https://doi.org/10.1016/j.autcon.2015.09.010

Tashakkori, H., Rajabifard, A. & Kalantari, M. (2015). A new 3D indoor/outdoor spatial model for indoor emergency response facilitation. *Building and Environment*, *89*, 170–182. https://doi.org/10.1016/j.buildenv.2015.02.036

Thomas, D. & Open Robotics. (2017a, 28. Juni). *Changes between ROS 1 and ROS 2.* Open Robotics. https://design.ros2.org/articles/changes.html (Zuletzt abgerufen am 21.07.2021).

Thomas, D. & Open Robotics. (2017b, 20. September). *ROS 2 middleware interface.* Open Robotics. https://design.ros2.org/articles/ros_middleware_interface.html (Zuletzt abgerufen am 21.07.2021).

Thrun, S. (2002). Particle Filters in Robotics. In A. Darwiche & N. Friedman (Hrsg.), *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence* (S. 511–518). Morgan Kaufmann. http://robots.stanford.edu/papers/thrun.pf-in-robotics-uai02.pdf

Tittmann, P. (2019). *Graphentheorie: Eine anwendungsorientierte Einführung* (3., aktualisierte Auflage). Carl Hanser Verlag.

Turau, V. (2009). *Algorithmische Graphentheorie* (3., überarb. Aufl.). Oldenbourg Verlag.

van Strien, E. (2015). *Using IfcOpenshell and pythonOCC to generate cross sections directly from an IFC file*. https://academy.ifcopenshell.org/posts/using-ifcopenshell-and-pythonocc-to-generate-cross-sections-directly-from-an-ifc-file/ (Zuletzt abgerufen am 02.09.2021).

VDI Verein Deutscher Ingenieure e.V. (2020-07). *Building Information Modeling - Grundlagen: Blatt 1* (VDI 2552:2020-07). Berlin. Beuth-Verlag GmbH. https://www.vdi.de/richtlinien/details/vdi-2552-blatt-1-building-information-modeling-grundlagen

Voigt, E. & Kremsreiter, M. (2020). *LiDAR in Anwendung*. https://www.tu-chemnitz.de/physik/EXSE/ForPhySe/LiDAR_in_Anwendung.pdf (Zuletzt abgerufen am 29.06.2021).

Voronoi, G. (1908). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik (Crelles Journal)*(133), 97–102. https://doi.org/10.1515/crll.1908.133.97

Wei, Y. & Akinci, B. (2018). End-to-end Image-based Indoor Localization for Facility Operation and Management. In J. Teizer & M. König (Hrsg.), *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC), Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC).* International Association for Automation and Robotics in Construction (IAARC). https://doi.org/10.22260/ISARC2018/0156

Woodside Capital Partners & Yole Développement. (2018). *The Automotive LiDAR Market*. https://cloudfront.net/production/onboardings/5e5421415aaa397b552399b4/documents/file/Yole_WCP-LiDAR-Report_April-2018-FINAL.pdf (Zuletzt abgerufen am 20.06.2021).

Yuan, W. & Schneider, M. (2010). Supporting 3D route planning in indoor space based on the LEGO representation. In S. Winter, C. S. Jensen & K.-J. Li (Hrsg.), *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness - ISA '10* (S. 16). ACM Press. https://doi.org/10.1145/1865885.1865890

VII ANLAGENVERZEICHNIS

ANLAGE 1 DIGITALES ANLAGENVERZEICHNIS

Dieses digitale Anlagenverzeichnis listet alle im digitalen Anhang befindlichen Dateien auf. Diese sind auf der Compact Disc am Ende der Diplomarbeit gespeichert.

Ordner	**Inhalt**
Dokumentation	Enthält diese Dokumentation als Word- und PDF-Datei sowie die Bibliographie als Projektdatei für Citavi 6 und BibTex
GenerateSections	Enthält das Skript generate_sections.py sowie ein Bild des Grundrisses des ersten Beispielgebäudes
IfcFile	Enthält die IFC-Datei des ersten Beispielgebäudes
makeROSmap	Enthält das Skript makeROSmap.py zur Erzeugung von Karten für ROS aus einer Bilddatei
rosPackage	Enthält in den Ordnern models und maps die erzeugten Simulationsmodelle als Mesh sowie die Karten der Gebäude

ANLAGE 2 QUELLTEXT: GENERATE_SECTIONS.PY

```python
import os

import OCC.Core.gp
import OCC.Core.Geom
import OCC.Core.Utils
import OCC.Core.Bnd
import OCC.Core.BRepBndLib
import OCC.Core.BRep
import OCC.Core.BRepPrimAPI
import OCC.Core.BRepAlgoAPI
import OCC.Core.BRepBuilderAPI
import OCC.Core.BRepAlgo
import OCC.Core.TopOpeBRepTool
import OCC.Core.ShapeExtend
import OCC.Core.GProp
import OCC.Core.BRepGProp
import OCC.Core.GC
import OCC.Core.ShapeAnalysis
import OCC.Core.TopTools
from OCC.Core.TopoDS import TopoDS

import ifcopenshell
import ifcopenshell.geom

# Specify to return pythonOCC shapes from ifcopenshell.geom.create_shape()
settings = ifcopenshell.geom.settings()
settings.set(settings.USE_PYTHON_OPENCASCADE, True)

# Initialize a graphical display window
occ_display = ifcopenshell.geom.utils.initialize_display()
occ_display.View.SetBackgroundImage("white_bg.bmp")

# Open the IFC file using IfcOpenShell
ifc_file = ifcopenshell.open(os.path.join(os.path.dirname(__file__), "<IFC-File>.ifc"))

# The geometric elements in an IFC file are the IfcProduct elements. So these are
# opened and displayed.
products = ifc_file.by_type("IfcProduct")
product_shapes = []

# For every product a shape is created if the shape has a Representation.
for product in products:
    if product.is_a("IfcOpeningElement") or product.is_a("IfcSite"): continue
    if product.Representation is not None:
        shape = ifcopenshell.geom.create_shape(settings, product).geometry
        product_shapes.append((product, shape))


# In this part the sections are created. You can enter the starting height, the
# maximum height and the height difference between each section.
starting_height = 3
maximum_height = 4
height_step = 3

section_height = starting_height
while section_height <= maximum_height:
    print("Section height          =", section_height)

    # A horizontal plane is created from which a face is constructed to intersect with
    # the building. The face is transparently displayed along with the building.
    section_plane = OCC.Core.gp.gp_Pln(
        OCC.Core.gp.gp_Pnt(0, 0, section_height),
```

```python
        OCC.Core.gp.gp_Dir(0, 0, 1)
    )
    section_face = OCC.Core.BRepBuilderAPI.BRepBuilderAPI_MakeFace(section_plane,
    -10, 10, -10, 10).Face()

    section_face_display = ifcopenshell.geom.utils.display_shape(section_face)
    ifcopenshell.geom.utils.set_shape_transparency(section_face_display, 0.5)
    for shape in product_shapes: ifcopenshell.geom.utils.display_shape(shape[1])

    raw_input()
    occ_display.EraseAll()


    # Each product of the building is intersected with the horizontal face
    for product, shape in product_shapes:
        section = OCC.Core.BRepAlgoAPI.BRepAlgoAPI_Section(section_face, shape).Shape()

        # The edges of the intersection are stored in a list
        section_edges = list(OCC.Core.Utils.Topo(section).edges())

        # If the length of the section_edges list is greater than 0 there is an
        # intersection between the plane (at current height) and the product. Only in that
        # case the product needs to be printed.
        if len(section_edges) > 0:
            print("    {:<20}: {}".format(product.is_a(), product.Name))

            # Open Cascade has a function to turn loose unconnected edges into a list of
            # connected wires. This function takes handles (pointers) to Open Cascade's native
            # sequence type. Hence, two sequences and handles, one for the input, one for the
            # output, are created.
            edges = OCC.Core.TopTools.TopTools_HSequenceOfShape()
            edges_handle = OCC.Core.TopTools.Handle_TopTools_HSequenceOfShape(edges)

            wires = OCC.Core.TopTools.TopTools_HSequenceOfShape()
            wires_handle = OCC.Core.TopTools.Handle_TopTools_HSequenceOfShape(wires)

            # The edges are copied to the sequence
            for edge in section_edges: edges.Append(edge)

            # A wire is formed by connecting the edges
            OCC.Core.ShapeAnalysis.ShapeAnalysis_FreeBounds.ConnectEdgesToWires(edges_handle,
            1e-5, True, wires_handle)
            wires = wires_handle.GetObject()

            # From each wire a face is created
            print("        number of faces = %d" % wires.Length())
            for i in range(wires.Length()):
                wire_shape = wires.Value(i+1)
                wire = TopoDS.wire(wire_shape)
                face = OCC.Core.BRepBuilderAPI.BRepBuilderAPI_MakeFace(wire).Face()

                # The wires and the faces are displayed
                ifcopenshell.geom.utils.display_shape(wire)
                face_display = ifcopenshell.geom.utils.display_shape(face)
                ifcopenshell.geom.utils.set_shape_transparency(face_display, 0.5)

                # Data about the wire is created to calculate the area
                wire_data = OCC.Core.ShapeExtend.ShapeExtend_WireData(wire, True, True)
                wire_data_handle = OCC.Core.ShapeExtend.Handle_ShapeExtend_WireData(wire_data)

    raw_input()
    occ_display.EraseAll()
```

ANLAGE 3 QUELLTEXT: MAKE_ROS_MAP.PY

```python
import cv2
import math
import os.path
import argparse
import time
import subprocess

def parsing():
    parser = argparse.ArgumentParser(description='Convert color image to ROS map (.pgm + .yaml).')
    parser.add_argument('image_file',
                        help='Color image to convert')
    parser.add_argument('-o', '--map_name',
                        help='Output map name',
                        default='map-ros')
    args = parser.parse_args()

    return args


def file_input_check(file_name):
    is_file = os.path.isfile(file_name)
    while not is_file:
        print("file '{}' doesn't exist. file name:".format(file_name))
        file_name = input('> ')
        is_file = os.path.isfile(file_name)

    file_size_kb = os.path.getsize(file_name) / 1024

    return file_name, file_size_kb


def read_color_image(file_name):
    img = cv2.imread(file_name)

    return img


def convert_to_binary(img, output_name):
    # convert to binary
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, bw_img = cv2.threshold(img, 220, 255, cv2.THRESH_BINARY)

    # Save image
    cv2.imwrite(output_name, bw_img)
    print("Saved binary image as '{}'\nin {}\n".format(output_name, os.getcwd()))

    return bw_img


def create_ROS_map(image, output_name):
    print("You will need to choose the x coordinates horizontal with respect to each other")
    print("Double Click the first x point to scale")
    print("(Press ESC to abort)")
    #
    # Some variables
    #
    ix, iy = -1, -1
    x1 = [0, 0, 0, 0]
    y1 = [0, 0, 0, 0]
    font = cv2.FONT_HERSHEY_SIMPLEX
    #
```

```
        # mouse callback function
        # This allows me to point and
        # it prompts me from the command line
        #
        def draw_point(event, x, y, flags, param):
            global ix, iy, sx, sy
            if event == cv2.EVENT_LBUTTONDBLCLK:
                ix, iy = x, y
                print(ix, iy)
                #
                # This is for the 4 mouse clicks and the x and y lengths
                #
                if x1[0] == 0:
                    x1[0] = ix
                    y1[0] = iy
                    print('Double click a second x point')
                elif (x1[0] != 0 and x1[1] == 0):
                    x1[1] = ix
                    y1[1] = iy
                    prompt = '> '
                    print("What is the x distance in meters between the 2 points?")
                    while True:
                        inp = input(prompt)
                        try:
                            deltax = float(inp)
                        except Exception as e:
                            pass
                        else:
                            break
                    dx = math.sqrt((x1[1] - x1[0]) ** 2 + (y1[1] - y1[0]) ** 2) * .01
                    sx = deltax / dx
                    print("You will need to choose the y coordinates vertical with respect to each other")
                    print('Double Click a y point')
                elif (x1[1] != 0 and x1[2] == 0):
                    x1[2] = ix
                    y1[2] = iy
                    print('Double click a second y point')
                else:
                    prompt = '> '
                    print("What is the y distance in meters between the 2 points?")
                    while True:
                        inp = input(prompt)
                        try:
                            deltay = float(inp)
                        except Exception as e:
                            pass
                        else:
                            break
                    x1[3] = ix
                    y1[3] = iy
                    dy = math.sqrt((x1[3] - x1[2]) ** 2 + (y1[3] - y1[2]) ** 2) * .01
                    sy = deltay / dy
                    print("Scale x: {:.5f}\nScale y: {:.5f}".format(sx, sy))
                    res = cv2.resize(image, None, fx=sx, fy=sy, interpolation=cv2.INTER_AREA)
                    res = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
                    tresh, res = cv2.threshold(res, 220, 255, cv2.THRESH_BINARY)

                    timestr = time.strftime("%Y%m%d-%H%M%S")
                    mapName = output_name

                    mapLocation = "map_" + timestr
                    os.mkdir(mapLocation)
                    completeFileNameMap = os.path.join(mapLocation, mapName + ".png")
                    completeFileNameYaml = os.path.join(mapLocation, mapName + ".yaml")
                    cv2.imwrite(completeFileNameMap, res)

                    print("\nConverting to .pgm ...")
                    cmd = "convert " + completeFileNameMap + " -compress none -depth 8 " + mapLocation + \
                          "/" + mapName + ".pgm"
                    cwd = os.getcwd()#os.path.dirname(os.path.realpath(__file__))
```

```
                subprocess.call(cmd, shell=True, cwd=cwd)
                print("Done.\n")

                with open(completeFileNameYaml, "w") as yaml:
                    #
                    # Write some information into the file
                    #
                    yaml.write("image: " + mapName + ".pgm\n")
                    yaml.write("resolution: 0.010000\n")
                    yaml.write("origin: [" + str(-1) + "," + str(-1) + ", 0.000000]\n")
                    yaml.write("negate: 0\noccupied_thresh: 0.65\nfree_thresh: 0.196")
                print("Saved map to '{}'".format(mapLocation))
                exit()

    cv2.namedWindow('image', cv2.WINDOW_NORMAL)
    cv2.setMouseCallback('image', draw_point)
    #
    #  Waiting for a Esc hit to quit and close everything
    #
    while 1:
        cv2.imshow('image', image)
        k = cv2.waitKey(20) & 0xFF
        if k == 27:
            break
        elif k == ord('a'):
            print('Done')
    cv2.destroyAllWindows()



def main():
    args = parsing()

    output_name = args.map_name
    file = args.image_file
    file_name, file_size_kb = file_input_check(file)

    img = read_color_image(file_name)

    print("\nConverting '{}' (size: {:.2f} kB) to binary image...".format(file_name, file_size_kb))
    convert_to_binary(img, "binary_output.png")

    binary_image = cv2.imread("binary_output.png")

    create_ROS_map(binary_image, output_name)


if __name__ == '__main__':
    main()
```