TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Bauingenieurwesen Institut für Bauinformatik

DIPLOMARBEIT

Effizienzsteigerung der Logistik auf Flachdächern mittels Simultaneous Localization and Mapping (SLAM) zur Steuerung von Robotern mit ROS

eingereicht von
cand. ing. Xiangyu Wang
geb. am 20.03.1996 in Beijing
Matrikel-Nummer: 4905714

Betreuer/in:

- Prof. Dr.-Ing. habil. Karsten Menzel / Prof. Dr.-Ing. Frank Will
- Dipl.-Ing. Shaowen Han

Dresden, den 26.09.2022

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: Wang

Vorname: Xiangyu

Matrikelnummer: 4905714

Dresden, den ABGABEDATUM

Unterschrift cand. ing. X. Wang

I Abstract

Die vorliegende Diplomarbeit beschäftigt sich mit dem Einsatz von Robotern, die mit Hilfe von SLAM die Baueffizienz von Flachdächern verbessern können. Die Arbeit kombiniert ein BIM-Modell mit einer Robotersimulationsumgebung, in der der Roboter 1:1 SLAM auf einem Gebäudemodell durchführen kann. Außerdem wird ein Verlegeplan auf der Grundlage der Abmessungen der Grundriss entwickelt und die Koordinaten der Zielpunkte und ihre Reihenfolge werden für das autonome Fahren und Navigieren des Roboters berechnet. Es wurde auch entwickelt, um die Dicke der Dämmplatten des Daches mit Hilfe von Computer Vision zu messen und so festzustellen, ob sie gegriffen werden sollten oder nicht. Die schon verlegten Dämmplatten wurden auch auf Fugen zwischen den verlegten Dämmplatten geprüft, um sicherzustellen, dass sie sich in der richtigen Position für den Verlegen befanden. Schließlich wurden zwei Simulationen und eine Feldimplementierung durchgeführt, um die Möglichkeit zu demonstrieren, BIM und Robotik bei der Verlegung von Flachdächern zu kombinieren und SLAM und Navigation zu nutzen, um autonom zu fahren und temporären Hindernissen auf der Baustelle auszuweichen.

II Inhaltsverzeichnis

III Abbildungsverzeichnis

IV Tabellenverzeichnis

V Abkürzungsverzeichnis

2D	Zweidimensional
3D	Dreidimensional
AMCL	Adaptive Monte Carlo Localization
BIM	Building Information Modeling
DIN	Deutsches Institut für Normierung
EPS	Expandierter Polystyrol Schaum Gegebenenfalls
LDS	Laser Distance Sensors
IFC	Industrie Foundation Classes
LIDAR	Light Detection And Ranging
IP	Internet Protocol
ISO	International Organization for Standardization
MCL	Monte Carlo Localization
OCC	Occupied
OGM	Occupancy Grid Map
PU	Polyurethan
RBPF	Rao-Blackwellized Partikelfilter
RGB	Red-Green-Blue
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SSH	Secure Shell
TB3	Turtlebot3
TF	Transform
URI	Uniform Resource Identifier
XPS	Extrudierter Polystyrol-Hartschaum

1 Einleitung

1.1. Motivation und Problemstellung

Mit dem Prozess der Digitalisierung und Automatisierung werden in immer mehr Branchen Roboter als Hauptproduktionsmittel eingesetzt. Allerdings ist der Baubereich, auf den über 10 % des deutschen Bruttoinlandsprodukts entfallen (), in Bezug auf die Digitalisierung schon immer unterdurchschnittlich gewesen (Beck 2022). Aufgrund des Ausbruchs von Covid-19 und anderer Faktoren kommt es auf dem deutschen Baumarkt häufig zu einem Arbeitskräftemangel und steigenden Personalkosten (Van Ooyen 2021, Apel 2022).

In den letzten Jahren wurde der Einsatz von Robotern im Bauwesen immer stärker erforscht, und angesichts der Vorteile, die sich aus der Verkürzung der Bauzeit, der Verbesserung der Qualität und der Erhöhung der Sicherheit auf der Baustelle ergeben, ist die Einführung von Robotern im Bauwesen sinnvoll. Deshalb ist es notwendig, ein entsprechendes Programm und Modell für den Einsatz von Robotern auf Baustellen zu entwickeln.

In diesem Fall stellt sich eine Frage. Kann der Arbeitskräftemangel durch den Einsatz von Robotern auf Baustellen gemildert werden, und kann die Bauindustrie noch stärker auf die Digitalisierung und Automatisierung ausgerichtet werden? Um diese Frage zu beantworten, wird in der Arbeit mit Hilfe von Simulationen überprüft, ob dies möglich ist.

Ziel dieser Arbeit ist es, ein Programm für einen Logistikroboter unter Verwendung von ROS zu entwickeln, um die Effizienz beim Verlegen von Dämmplatten auf Flachdächern zu erhöhen. Dabei wird mit Hilfe des BIM-Modells eine Simulationsumgebung für den Roboter entwickelt, die sowohl den Verlegeplan als auch Hindernisse mit einbezieht. Der Roboter kann auch selbstständig zu den festgelegten Zielpunkten fahren. Schließlich sollte der Roboter in der Lage sein, anhand von Fotos zu erkennen, wie dick die Dämmplatte ist und ob sie korrekt verlegt wurde.

1.2. Aufbau der Arbeit

In Kapitel 3 wird über die Methodik detailliert analysiert, wie BIM-Modelle

schrittweise in eine simulierte Umgebung umgewandelt werden können, die von Robotern genutzt werden kann, sowie die theoretischen Grundlagen und Algorithmen für SLAM und Wegplanung, und im folgenden Schritt wird erläutert, wie das ROS-System genutzt werden kann, um Roboter in die Lage zu versetzen, SLAM- und Wegplanungsalgorithmen zu nutzen, um die vorgegebenen Ziele des Roboters zu erreichen. Am Ende der Methodik wird die theoretische Grundlage des Canny-Operators vorgestellt, der im Bereich der Computer Vision verwendet wird, sowie die Konzeption eines Experiments zur Bestimmung der Schwellenwerte des Canny-Operators für diese Arbeit.

In Kapitel 4 wird die erwähnte Methode angewandt, um zwei Simulationsexperimente zu entwerfen und durchzuführen. In einem wird der Roboter als Logistikroboter auf einem Dach in einer Simulationsumgebung eingesetzt und bewältigt die Aufgabe. Dazu wird auf der Grundlage des BIM-Modells eine Simulationsumgebung entwickelt und ein Verlegeplan für das Modell berechnet. Danach werden SLAM und Navigation eingesetzt, damit der Roboter reibungslos zu den Zielpunkten im Verlegeplan fahren kann. Wenn der Roboter den Zielpunkt erreicht hat, misst er je nach Bedarf die Länge der Dämmplatten oder prüft mit Hilfe von Computer Vision, ob sie korrekt installiert sind. Nach der ersten Simulation wird am Institut für Bauinformatik in Nürnberg 31a eine zweite Simulationsumgebung entwickelt, in der der Roboter nur das SLAM in der Simulationsumgebung durchführt und dann den realen TB3 nutzt, um in der realen Umgebung anhand einer aus dem SLAM in der Simulationsumgebung abgeleiteten Gitterkarte zu navigieren, das Ziel durch Wegplanung zu erreichen und Aufgaben wie das Ausweichen vor Hindernissen zu erfüllen. Mit dieser Simulation soll nachgewiesen werden, dass mit Hilfe von BIM die simulierte Umgebung mit der realen Umgebung kombiniert werden kann, um die Effizienz zu steigern und die Schwierigkeiten von SLAM zu verringern.

In Kapitel 5 werden die aus den beiden Simulationen gezogenen Ergebnisse zusammengefasst. Das Projekt wird weiterhin am Institut für Bauinformatik durchgeführt, wobei die durch SLAM erhaltene Karte aus der vorherigen Simulation zusammen mit dem Verlegeplan aus der ersten Simulation verwendet wird, um einige Zielpunkte für den Roboter festzulegen. Der Roboter wird die Aufgabe erfüllen, zu den angegebenen Koordinaten zu fahren und dabei vorübergehenden Hindernissen in einer realistischen Umgebung auszuweichen, und er soll auch die Aufgabe der visuellen Erkennung erfüllen können, sobald er den Zielpunkt erreicht hat. Zum Schluss der Arbeit gibt es eine Zusammenfassung und einen Ausblick auf die künftige Forschung.

2 Stand der Technik

Der erfolgreiche Einsatz von Robotern für Bauarbeiten auf Baustellen erfordert die Zusammenarbeit verschiedener Bereiche. Im Rahmen früherer Untersuchungen wurde in den verschiedenen Bereichen bereits in Erfahrung und Wissen gebracht. Daher wird der aktuelle Stand der Technik in drei Bereichen untersucht: Flachdächer, Robotik und BIM.

2.1. Flachdach

Dächer sind ein Bestandteil der Schutzhülle von Gebäuden und werden seit langem durch die verfügbaren Baumaterialien und die Witterungsbedingungen bestimmt. Vor etwa 8.000 Jahren wurde bereits in einer Stadt der Türkei alle Häuser mit Flachdächern gebaut. Demgegenüber sind in Nord- und Mitteleuropa Steildächer seit Jahrhunderten die typische Form der Häuser. Nach der Entdeckung des Bitumens wurden die konstruktiven Fähigkeiten von Flachdächern erheblich verbessert und die Kosten gesenkt. Seit dieser Zeit werden in Europa Flachdächer in großer Zahl gebaut (Pech et al. 2011, S. 1).

Bis zu den 1960er Jahren waren Flachdächer in Deutschland nicht der Trend, sondern wurden nur bei Gewerbe- und Industriebauten verwendet, während im Wohnungsbau das Steildach noch absolut dominant war.[1] Mit dem Aufkommen der Moderne begann man in Deutschland jedoch, Wohnhäuser mit Flachdach zu bauen. Durch die Verwendung eines Flachdachs kann das Dach als Terrasse genutzt werden, wodurch sich die Größe der Gebäude erhöht und auch der Innenraum von Gebäuden vergrößert wird, solange keine Dachneigung vorhanden ist. Angesichts der zunehmenden Bedeutung des Umweltschutzes im Bauwesen werden weitere Vorteile von Flachdächern immer mehr zum Thema, wie z. B. die einfache Installation von Photovoltaikanlagen, die höhere Energieeffizienz und die Möglichkeit, Dachgärten anzulegen, um das Wohnumfeld zu verbessern und die städtische Grünfläche zu vergrößern. In den letzten Jahren sind Flachdächer in

[1] Vgl. https://dietrich-bedachung.de/2018/06/12/das-flachdach-modischer-trend-oder-echte-alternative/

Abb. 2.1 Flachdach in Deutschland
Quelle: Eigene Aufnahme bei Praktikum bei ArchifuStudio in Mönchengladbach

Deutschland wieder gefragt.[2] In Abbildung 2.1 ist ein Flachdach über einer Garage dargestellt.

Die bauliche Form eines Daches wird in der Regel durch die Dachneigung unterschieden: Dachneigungen von mehr als 20 Grad sind Steildächer, Dachneigungen zwischen 5 und 20 Grad sind flach geneigte Dächer und Dachneigungen von weniger als 5 Grad sind Flachdächer (Birkner und Willems 2013, S. 447). Die Mindestneigung eines Daches beträgt 2%, um eine reibungslose Entwässerung zu (DIN 18531-1, 2017, S. 14).

Die meisten verwendeten Fotos und Informationen zu den Dämmplatten wurden

[2] Vgl. https://www.myhomebook.de/projects/bauen/flachdach-hausbau-trend#h-flachdach-im-trend-die-vor-und-nachteile

von dem Partner in Dresden Dachdeckermeister Claus Dittrich GmbH & Co. aufgenommen.

2.1.1. Bauarten und Aufbau der Flachdächer

Nach DIN 18531-1 (2017-07, S. 7) werden Flachdächer grob in zwei Kategorien unterteilt. Eine Kategorie sind die nicht genutzten Dächer, das heißt Dächer, die außerhalb von Wartungs- und Instandhaltungsarbeiten nicht befahren werden oder extensiv begrünt sind. Die andere Kategorie sind nutzbare Dächer, das sind Dächer, die für den Menschen begehbar sind, wie z. B. Terrassen, Gehwege, Dächer mit Solaranlagen oder anderen haustechnischen Anlagen, oder Dächer mit intensiver Begrünung und weniger als 100 mm Anstaubewässerung.

Die beiden oben genannten Dacharten werden auch in belüftete Dächer (Kaltdächer) und nicht belüftete Dächer (Warmdächer) unterteilt, die davon abhängen, ob die Dachkonstruktion eine Luftschicht enthält oder nicht. Bei Kaltdächern sind die Abdichtungs- und die Dämmschicht durch eine Luftschicht getrennt, durch die der Wasserdampf abgeleitet wird und die Wärme über diesen Weg entweicht. Im Gegensatz dazu sind Warmdächer, bei dem die Abdichtung und die Dämmung miteinander verbunden sind. Die Wärme aus dem Inneren des Gebäudes diffundiert in unmittelbarem Kontakt mit der kalten Luft an die Außenseite des Daches, und der Wasserdampf kann nicht aus dem Haus entweichen (Hestermann und Rongen 2018, S. 284).

Ein komplettes Dach verfügt über verschiedene bauliche Schichten (Birkner und Willems 2013, S. 452), die unterschiedlichen Anforderungen erfüllen, z. B. Tragfähigkeit, Wasserdichtigkeit und Wärmeschutz. Im Folgenden werden die verschiedenen Bestandteile der Struktur und deren Funktion kurz beschrieben. Eine Tragkonstruktion ist die grundlegendste und wichtigste Komponente jeder baulichen Anlage, wobei die üblichen Dachtragwerke aus Ortbeton, Betonfertigteilen oder Stahltrapezprofilen aufgebaut sind. Nach dem Ausführen der Tragkonstruktion entstehen selbstverständlich Unebenheiten durch Schwinden des Baustoffs oder Spannungsrisse, daher ist eine Ausgleichsschicht erforderlich, um diese Unebenheiten auszugleichen und den weiteren Teil der Dächer vor chemischer Korrosion mit der darunter liegenden Konstruktion zu schützen. Als äußere Hülle des Gebäudes befindet sich das Dach natürlich in unmittelbarem Kontakt mit der äußeren Umgebung. Falls Regenwasser von außen in den Innenbereich des Gebäudes eindringt, führt dies zu Schimmelbildung in den

Räumen und zu Schäden an der Baukonstruktion und an den Möbeln. Umgekehrt entsteht im Inneren des Gebäudes auch Dampf, der sich über das Dach nach oben ausbreiten kann, wodurch die Möglichkeit besteht, dass Tauwasser in die Dämmung eindringt und die Wärmedämmung des Gebäudes beschädigt. Deswegen sind sowohl die Dampfsperre als auch die Wasserabdichtung wesentliche Bestandteile des Daches (Birkner und Willems 2013, S. 452 - 459). Im Hinblick auf die Nutzung des Gebäudes sind die zwei wichtigsten Faktoren die Wasserdichtigkeit und die Wärmeisolierung. Die Isolierung erfolgt durch Dämmungsplatten, die meistens aus EPS, XPS, Foamglas, Mineralwolle oder PU für Flachdächer hergestellt werden.[3] Die Dämmplatten werden in genormten Größen hergestellt und die Dicke der Dämmplatten wird nach DIN 4108-2 (2013-02, S. 15 - 23) bestimmt. Bei Flachdächern kann die Anforderung an die Mindestgefälle von 2% auch durch Gefälledämmung erfüllt werden. Dies wird im Folgenden im detaillierten analysiert werden.

2.1.2. Gefälle des Dachs

In der Abbildung 2.2 ist eine Detailzeichnung eines konventionellen nicht belüfteten Daches dargestellt. Auf der Stahlbetontragschicht liegt die Ausgleichsschicht, anschließend die Gefälledämmplatte und darauf die Abdichtung. Die Dachabdichtung schützt jedoch nur davor, dass Regenwasser zeitweilig in die übrige Dachkonstruktion oder in das Innere des Gebäudes eindringt, sie verfügt allerdings nicht über die Eigenschaft, das Regenwasser abzuleiten. Wenn das Regenwasser nicht rechtzeitig abfließt oder nicht abgeleitet werden kann, bildet es eine Wasserlache auf dem Dach.

Als Flachdach ist stehendes Wasser grundsätzlich schädlich für die Dachkonstruktion. Daher muss ein Flachdach nicht nur wasserdicht sein, sondern auch die Möglichkeit haben, Wasser vom Dach abzuleiten. Um sicherzustellen, dass das Regenwasser vollständig abfließen kann, muss die Oberfläche der

Abdichtungsschicht ein Mindestgefälle von 2 % aufweisen. (DIN 18531-1 2017, S.14). Gemäß DIN 1986-100 (2016 S. 28 - 30) sind Dachentwässerungsanlagen für die Ableitung des Niederschlagswassers auf dem kürzesten Weg einzusetzen, damit das Niederschlagswasser möglichst unverzögert und frühzeitig vom Dach abgeleitet

[3] Vgl. Kurze Metting mit J. Dittrich von Dachdeckermeister Claus Dittrich GmbH, am 18.07.2022

wird (Fachregel für Abdichtungen Flachdachrichtlinie 2007, S. 12) (Birkner und Willems 2013, S. 475).

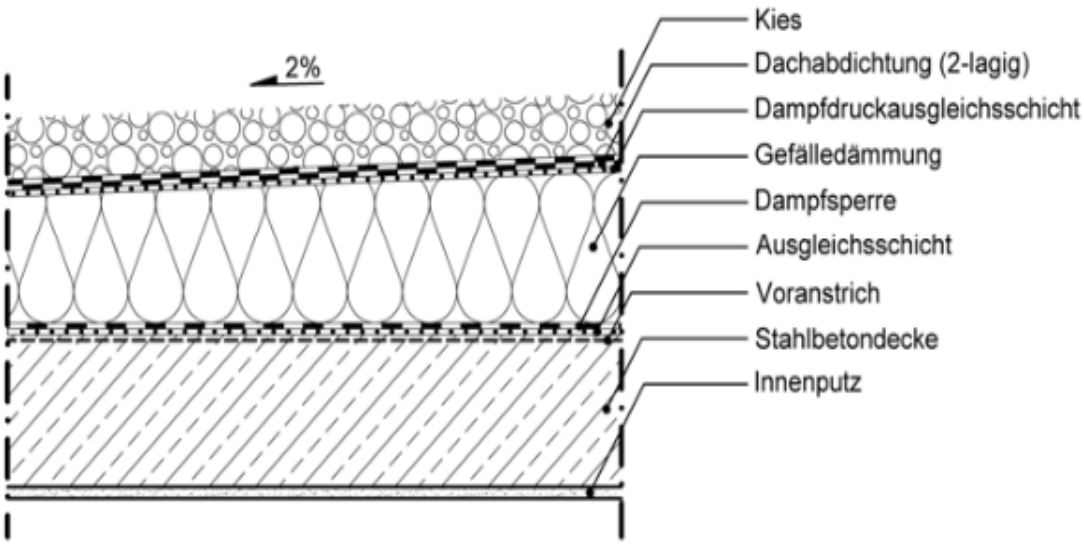

Abb. 2.3 Detailplan von konventionellen nicht belüfteten Flachdächern
Quelle: Birkner et al. 2013, S. 447

Wie in Abbildungen 2.3 und 2.4 dargestellt, sind die meisten Flachdächer mit einem internen Entwässerungssystem konzipiert (Hestermann und Rongen 2018, S. 332), wobei der Zulauf zur Innenentwässerung am tiefsten Punkt der Dachfläche und mit einem Notüberlauf am Rand des Daches konzipiert werden sollte. Damit soll sichergestellt werden, dass das Niederschlagswasser vollständig und reibungslos abfließen kann und das Risiko von Wasseransammlungen auf dem Dach vermieden wird (Birkner und Willems 2013, S. 475).

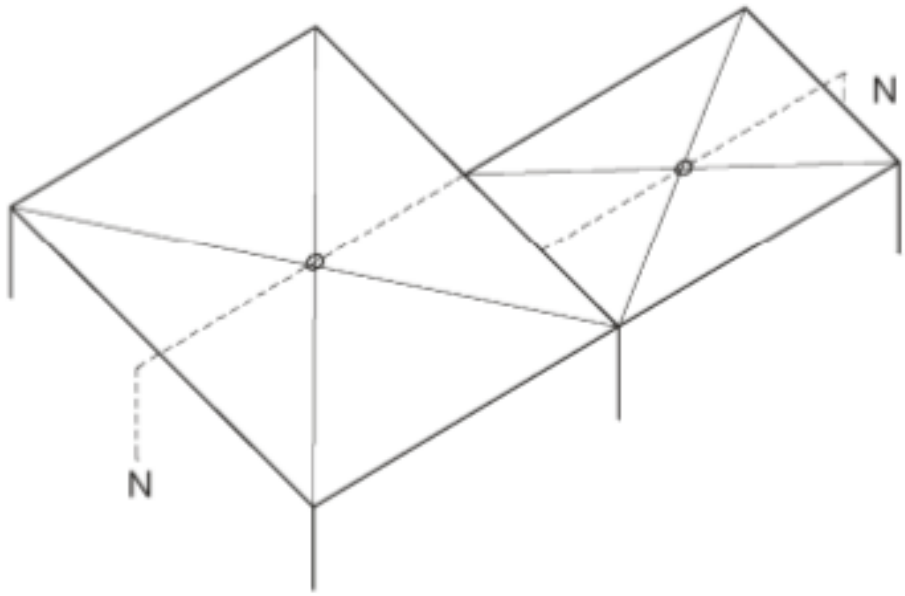

Abb. 2.3 Grundriss mit Lage der Entwässerungsstellen
Quelle: Herstermann et al. 2018, S. 333

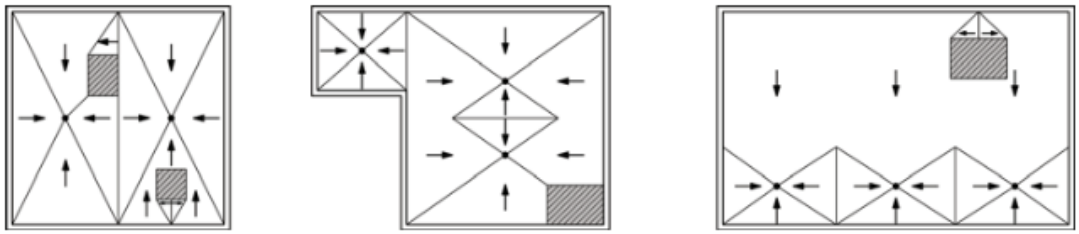

Abb. 2.4 Gefällefelder von Innenentwässerung der Flachdächer
Quelle: Pech et al. 2011, S. 25

Hierfür müssen Gefälle künstlich geschaffen werden, sodass Dachkehle gebildet werden, damit das Regenwasser reibungslos den niedrigsten Punkt der Dachfläche erreichen kann. Entweder mit Hilfe von Gefällebeton oder durch den Einsatz von geeignet zugeschnittenen Dämmplatten ist es möglich, in der Dachfläche ein den Anforderungen entsprechendes Gefälle zu schaffen (Hestermann und Rongen 2018, S. 332). In Abbildung 2.5 sind drei Möglichkeiten zur Schaffung von Gefällen dargestellt.

Der Einbau von Gefällebeton auf Betonplatten von Rohbau ist schon lange nicht mehr üblich. Das in der Abbildung auf der rechten Seite gezeigte Verfahren, bei dem die ursprünglich geplante Rohbaukonstruktion der Dachflächen geneigt sind, zurzeit werden die meisten Dächer mit Gefälledämmung ausgeführt (Hestermann und Rongen 2018, S. 332).

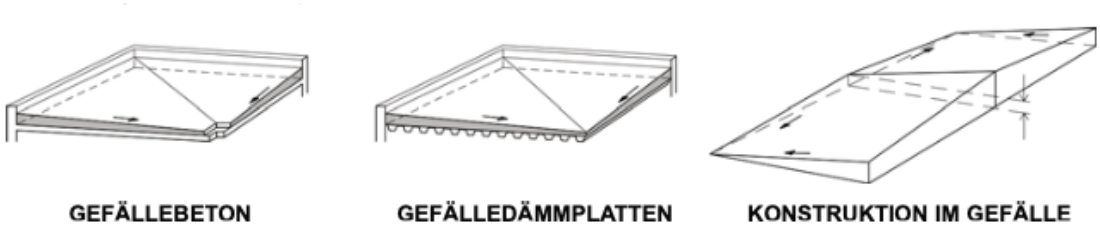

Abb. 2.5 Gefällefelder von Innenentwässerung der Flachdächer
Quelle: Pech et al. 2011, S. 25

2.2. Robotik im Bauwesen

Roboter im Bauwesen stellen eine Form der automatisierten Fertigung dar, die durch die Automatisierung einiger der wichtigsten Vorgänge die Kosten für diese Tätigkeiten senkt. Gebäude werden auch als Immobilien bezeichnet, und wegen des "unbeweglichen" Charakters des Bauens sind die meisten der im Bauwesen eingesetzten Roboter mobile oder bewegliche Systeme. Entweder geht es um die Bewegung als Teil des Arbeitsprozesses, wie z. B. robotergesteuerte Bulldozer oder Fliesenlegerroboter, oder um Roboter, die sich bewegen müssen, um ihren Arbeitsbereich zu erweitern, wie z. B. Mauerwerks- oder Farbspritzroboter. Entweder geht es um die Bewegung als Teil des Arbeitsprozesses, wie z. B. robotergesteuerte Bulldozer oder Fliesenlegerroboter, oder um Roboter, die sich bewegen müssen, um ihren Arbeitsbereich zu erweitern, wie z. B. Mauerwerks- oder Farbspritzroboter. Bauroboter werden je nach Einsatzort in zwei Hauptkategorien eingeteilt: Baustellenroboter, die bestimmte Aufgaben auf der Baustelle ausführen, und Roboter, die außerhalb der Baustelle in Fabriken vorgefertigte Bauteile herstellen (Saidi et al. 2016, S. 1496).

In den 1970er Jahren begann mit der fortschreitenden Automatisierung das Aufkommen von Robotern, die nur eine bestimmte Funktion erfüllen. Ihre Hauptaufgabe und ihr Ziel ist es, die Arbeiter bei der Durchführung eines bestimmten Bauprozesses oder einer bestimmten Aufgabe zu unterstützen, z. B. beim Betonieren, Streichen usw. Ihre Hauptaufgabe und ihr Ziel ist es, die Arbeiter bei der Durchführung eines bestimmten Bauprozesses oder einer bestimmten Aufgabe zu unterstützen, z. B. beim Betonieren, Streichen usw. Diese Aufgaben haben alle eine Eigenschaft, nämlich dass sie häufig wiederholende Arbeiten erfordern. Die meisten Roboter benötigen nur 1 oder 2 Personen, um sie zu bedienen, und sie erhöhen die Effizienz dieser Bauprozesse und gewährleisten die Sicherheit der Arbeiter. Aufgrund des unvorhersehbaren und dynamischen

Charakters von Baustellen können jedoch nur wenige Roboter effektiv und effizient eingesetzt werden. Obwohl die Effizienz des Betriebs verbessert und Arbeitskosten eingespart werden, führen die übermäßigen Beschränkungen und die Notwendigkeit, jede Baustelle einzeln zu transportieren, vorzubereiten und zu programmieren, nicht zu einer erheblichen Senkung der Gesamtkosten (Saidi et al. 2016, S. 1497).

Völlig autonome Konstruktionsroboter sind im Gegensatz dazu fähig, vorgegebene Aufgaben selbstständig und ohne menschliches Zutun auszuführen. Ein Konstruktionsroboter muss zunächst seine Umgebung wahrnehmen und sich an sie anpassen, die Ausführung seiner Aufgaben planen und sie gegebenenfalls neu planen. Intelligente Bauroboter sollten auch in der Lage sein, festzustellen, wann ihre Aufgaben nicht erfüllt werden können, und Unterstützung anfordern. Baustellen sind geprägt von unregelmäßiger Geometrie, zahlreichen Hindernissen usw., so dass die Forscher derzeit versuchen, den Robotern die Möglichkeit zu geben, ihre Umgebung zu kartieren und selbständig durch sie zu navigieren, was der Schwerpunkt dieser Arbeit ist (S.M.S. Elattar 2008, S. 24).

Der in dieser Arbeit verwendete Turtlebot3 wird im Folgenden beschrieben.

2.2.1. Modulare Turtlebot 3

Turtlebot stammt aus dem Jahr 1967 und ist ein ROS-Standard-Plattformroboter, der zurzeit in der dritten Generation entwickelt wird. Der modulare Roboter Turtlebot3 basiert auf dem Raspberry Pi und ist mit dem 360-Grad-LIDAR LDS-01 ausgestattet, das eine effektive Reichweite von 3,5 m hat (Robotis 2022a) und für laserbasiertes SLAM und Navigation verwendet werden kann. Auch mit einer 3D-Kamera kann visuelles SLAM durchgeführt werden. Es gibt drei offizielle Turtlebot3-Modelle, Turtlebot3 Burger, Waffle und WafflePi. Das in dieser Arbeit verwendete Modell ist der Turtlebot3 WafflePi, wobei sich Verweise auf "Roboter" und "Turtlebot3" im Folgenden auf den Turtlebot3 WafflePi beziehen, sofern nicht anders erwähnt (Pyo et al. 2017, S. 279 - S.283).

Turtlebot3 kann nicht nur in der Realität mit verschiedenen funktionalen Zubehörteilen ausgestattet werden, sondern auch in einer Simulationsumgebung durch Anpassung der Konfigurationsdateien mit verschiedenen Geräten belegt werden.

2.2.2. SLAM

Simultane Lokalisierung und Kartierung (SLAM) ist ein wesentliches Thema bei der Untersuchung mobiler Roboter. SLAM ist definiert als die gleichzeitige Schätzung der Position eines mobilen Roboters und die Erstellung einer Karte seiner Umgebung durch die Verarbeitung von Daten, die durch die Erkennung interner Zustände (Odometer, Gyroskope usw.) oder durch die Erfassung der äußeren Umgebung durch Lasersensoren, Bildsensoren usw. gewonnen werden. Außerdem besteht das Wesentliche der Objektverfolgung in der Lokalisierung eines beweglichen Zielobjektes (Pyo et al. 2017, S. 332) (Stachniss et al. 2016, S. 1154). SLAM kann in zwei Kategorien eingeteilt werden, nämlich in Laser-basiertes SLAM und Vision-basiertes SLAM, wobei die Art der externen Sensoren, die zur Identifizierung von Objekten in der Umgebung des Roboters verwendet werden, berücksichtigt wird (Gurel 2018).

Aktuelle SLAM-fähige mobile Roboter haben sich in einer Vielzahl von Szenarien bewährt. Gatesichapakorn et al. (2019) und An et al. (2021) haben 2D-Radar- oder RGB-Kamera-basiertes SLAM auf mobile Roboter angewandt, um autonomes Fahren in Innenräumen, Wegplanung über große Entfernungen und lange Zeiträume sowie die Implementierung von Autonome Hindernisvermeidung

2.2.3. OpenCV

OpenCV ist eine Open-Source-Bibliothek für Computer Vision. Die Bibliothek ist in C und C++ geschrieben und kann unter Linux, Windows und Mac OS X ausgeführt werden. Sie kann derzeit über eine Schnittstelle mit Python und Matlab gesteuert werden (OpenCV tram 2022). Eines der Ziele bei der Entwicklung von OpenCV war es, die Rechenleistung zu verbessern und sich auf Echtzeitanwendungen zu konzentrieren. Ein weiteres Ziel ist die Bereitstellung einer benutzerfreundlichen Bildverarbeitungsinfrastruktur für die Computervision, mit deren Hilfe relativ komplexe Bildverarbeitungs-anwendungen schnell erstellt werden können (Bradski und Kaehler 2008, S. 1).

Heutzutage bietet vision_opencv auch eine beträchtliche Anzahl von OpenCV-Bibliotheken, die für ROS verfügbar sind, und es ist nun möglich, OpenCV in ROS wie jedes andere Drittanbieter-Paket zu verwenden (Open Robotics 2020d). OpenCV ist auch sehr freundlich zu maschinellem Lernen, so dass ein sehr breites Spektrum an maschinellen Lerninhalten mit OpenCV durchgeführt werden kann (Bradski und

Kaehler 2008, S. 459 - 520).

2.2.4. Robot Operating System (ROS)

Neben der Programmierung und den Algorithmen ist auch das Betriebssystem für Roboter sehr wichtig. In der traditionellen Robotikindustrie verwenden die einzelnen Roboter verschiedener Hersteller unterschiedliche Betriebssysteme und Programmiersprachen und arbeiten nicht miteinander zusammen (Pyo et al. 2017, S. 5).

Das Robot Operating System, auch ROS abgekürzt, ist ein plattformübergreifendes, Open Source Meta-Betriebssystem für Roboter, welches eine Hardware-Abstraktionsschicht, die zugrundeliegende Gerätesteuerung, die Implementierung gemeinsamer Funktionen, die prozessübergreifende Nachrichtenübermittlung und die Verwaltung von Softwarepaketen umfasst. ROS entstand am Robotics Lab der Stanford University als ein Programm zur Entwicklung eines Roboters mit künstlicher Intelligenz. Im November 2007 übernahm das Robotikunternehmen Wiilow Garage die Entwicklung von ROS. Die endgültige Version 1.0 von ROS wurde offiziell im Jahr 2010 veröffentlicht (Pyo et al. 2017, S. 15).

ROS ähnelt architektonisch einem Computerbetriebssystem und führt Prozesse wie Zeitplanung, Laden, Überwachung und Fehlerbehandlung durch, indem es eine Virtualisierungsschicht zwischen der Anwendungssoftware und den verteilten Rechenressourcen verwendet (Pyo et al. 2017, S. 11). Sie bietet eine Vielzahl von Entwicklungs-umgebungen sowie eine Bibliothek mit fertigen Tools und Paketen, die direkt aus der Bibliothek heraus installiert und verwendet werden können. Es ist auch möglich, dass Benutzer ihre eigenen Programme schreiben und ihre eigenen Pakete erstellen. Es ist ein sprachunabhängiges Betriebssystem und kann in einer Vielzahl von modernen Programmsprachen, wie C++, Python usw., geschrieben werden. ROS unterscheidet sich von traditionellen Computerbetriebssystemen jedoch dadurch, dass für den Betrieb von ROS zunächst ein Computerbetriebssystem installiert werden muss. ROS läuft derzeit nur auf Unix-basierten Plattformen wie Ubuntu, einer Linux-Distribution (Joseph 2018, S. 8–9).

Die Architektur von ROS ist in drei Schichten unterteilt: Die erste Schicht ist die Dateisystemschicht, in der die Ordnerstruktur von ROS, der interne Aufbau und die Mindestanzahl von Dateien zur Aufrechterhaltung der Arbeit von ROS enthalten sind. Die zweite Schicht ist die Computergraph-Schicht, deren Hauptzweck in der Kommunikation zwischen Prozessen und Systemen besteht. Die dritte Ebene ist die

Community-Ebene, auf der, wie bei den meisten Open-Source-Projekten, Wissen, Algorithmen und Code zwischen den Entwicklern ausgetauscht werden (Anil Mahtani et al. 2016 S. 39).

2.3. Building Information Modelling

Die Planung und Ausführung eines Gebäudes ist ein komplexer Prozess, der die Unterstützung einer großen Anzahl von Teilnehmern aus verschiedenen Fachbereichen erfordert. Gegenwärtig erfolgt der Informationsaustausch zwischen den an Bauprojekten Beschäftigten hauptsächlich auf der Grundlage von Zeichnungen, auch der Einsatz von 2D-Zeichensoftware wie Auto CAD ist eine einfache Nachahmung einer Arbeitsmethode, die seit Jahrhunderten existiert (Borrmann et al. 2021, S. 2). Aus diesem Grund werden zwar auch digitale Werkzeuge für die Planung, Ausführung und den Betrieb von Gebäuden eingesetzt, aber vor allem durch die Verwendung von Konstruktionszeichnungen werden die Informationen über das Gebäude nicht vollständig zwischen den verschiedenen am Bau beschäftigten Personen genutzt. Das bedeutet, dass jeder am Bau Beteiligte für sich allein arbeitet und dass es keine angemessene Koordinierung der Unterlagen untereinander gibt, bevor mit der Ausführung des Gebäudes begonnen wird, was zu vielen Problemen führen kann, von denen das Schlimmste darin besteht, dass viele sehr folgenschwere Fehler erst während der Ausführung des Baus entdeckt werden können (Spengler und Peter 2020, S. 2–3). Außerdem kann die Zeichnung als Strichzeichnung nicht von einem Computer analysiert werden, und alle Informationen, die sie über das Gebäude enthält, können nicht mit dem Computer digital entwickelt und verarbeitet werden. Deshalb bleibt das enorme Potenzial, das die Möglichkeiten der Informationstechnologie im gesamten Lebenszyklus eines Gebäudes bieten, praktisch ungenutzt (Beetz et al. 2018, S. 2).

Der gesamte Lebenszyklus eines Gebäudes ist eine hochgradig interdisziplinäre Aufgabe. Die beste Lösung besteht daher darin, ein System zu schaffen, das den gesamten Lebenszyklus eines Gebäudes abdeckt, eine nahtlose digitale Prozesskette, über die alle Teilnehmer Informationen austauschen können, integrierte digitale Gebäudemodelle zu verwenden und auf offene, einheitliche Standards und eine geeignete gemeinsame Datenumgebung für das Datenmanagement zu setzen (Spengler und Peter 2020, S. 3).

Das Konzept ist eigentlich nicht neu, bereits 1974 präsentierten Eastman et al. einen Papierbeitrag über die Entwicklung und Nutzung von virtuellen Gebäudemodellen

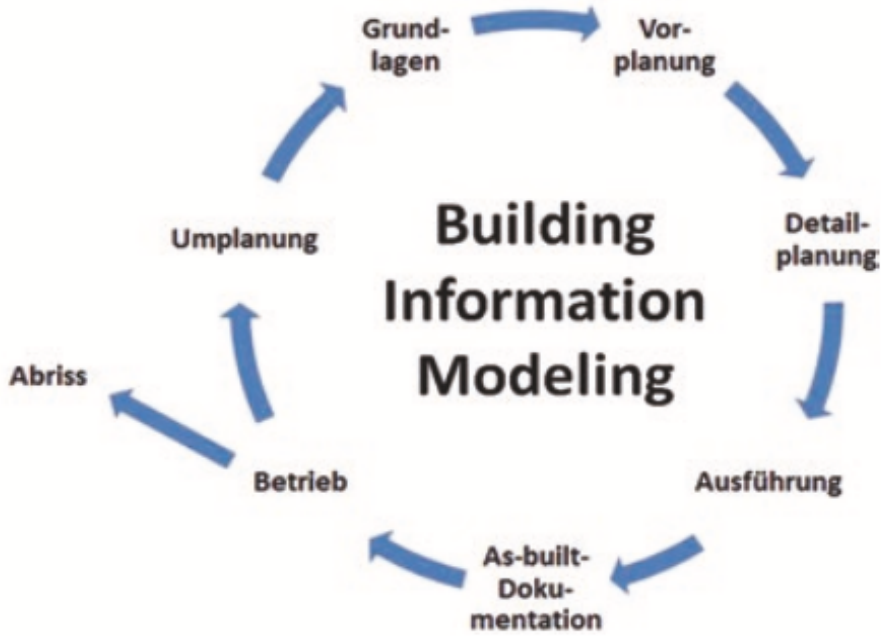

Abb. 2.6 Lebenszyklus eines Gebäudes mit BIM
Quelle: Blankenbach et al. 2020, S. 779

und das Konzept Building Information Modelling, kurz BIM, definierten.

Das offensichtlichste Merkmal eines Gebäudeinformationsmodells ist die dreidimensionale Geometrie des Gebäudes, ein umfassendes digitales Bild, welches außerdem in der Regel nicht-physikalische Objekte wie Raum- und Flächeneinteilungen sowie eine geschichtete Projektstruktur enthält (Beetz et al. 2018, S. 4) (Blankenbach und Becker 2020, S. 784) und die Ableitung konsistenter zweidimensionaler Grundrisse und Querschnitte ermöglicht. Es reicht jedoch nicht aus, nur 3D-Modelldaten eines Gebäudes zu haben. Die in BIM enthaltenen Objekte sind mit einer genau definierten Semantik verknüpft, d. h. mit weiteren Informationen über die Beschreibung und die Eigenschaften des Objekts (Borrmann et al. 2021, S. 6).

Mit Hilfe von BIM können die Gebäudedaten über den gesamten Lebenszyklus (siehe Abbildung 2.6) eines Gebäudes genutzt werden, was nicht nur die Qualität der Planung, sondern auch die Effizienz von Baudurchführungen verbessert. Auf diese Weise können Fehler, die früher sehr einfach und kostspielig zu erscheint waren, noch vor Beginn der Ausführungsphase erkannt und behoben werden. In diesem Zusammenhang liegt das große Potenzial von BIM. Darüber hinaus bieten exakte 3D-Modelle mit klar definierter semantischer Modellierung großes Potenzial und Möglichkeiten für den Einsatz von Gebäudeautomation und Robotik im Bauwesen (Borrmann et al. 2021, S. 6).

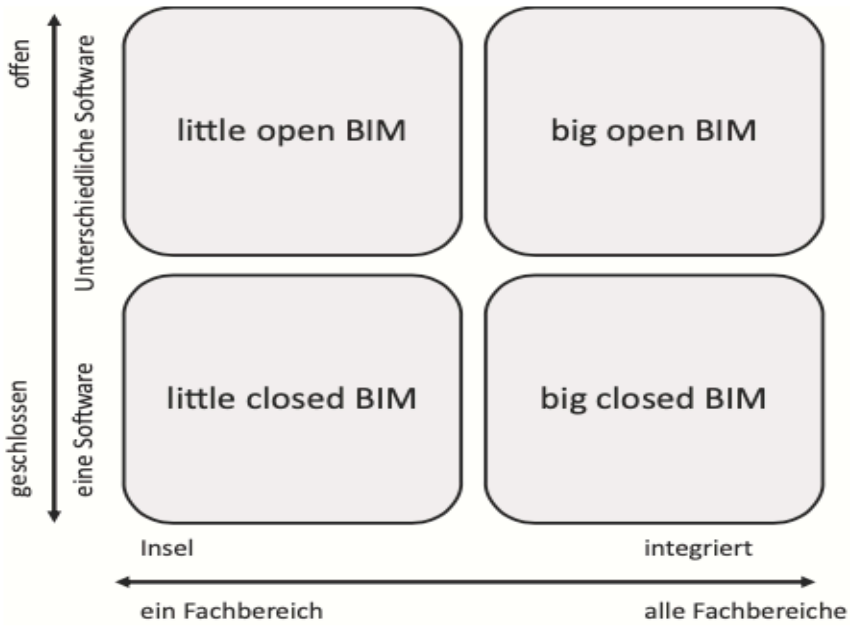

Abb. 2.7 Small/Big, Closed/Open BIM
Quelle: Spengler et al. 2020, S. 13

2.3.1. Big/Small, Open/Close BIM

Heute wird BIM in Big BIM, Small BIM, Open BIM und Closed BIM unterteilt (Blankenbach und Becker 2020, S. 783). Wenn es nur Architekten gibt, der eine bestimmte BIM-Software in seinem Fachgebiet einsetzt, spricht man von Small BIM. Mit der Software werden digitale Gebäudemodelle generiert, auf deren Grundlage Pläne entwickelt werden (Gregor Fleischmann 2021, S. 243). Das Modell wird jedoch nicht zur Koordinierung der Planung zwischen den verschiedenen Fachbereichen eingesetzt, da das Modellwissen als isolierte Fachlösung verwendet wird und die Kommunikation mit anderen Beteiligten nach wie vor auf Zeichnungen beruht. Der Einsatz von BIM erhöht lediglich die Effizienz der Architekten in der Entwurfsphase, und das Potenzial der digitalen Informationsmegalopolis des Gebäudes wird noch nicht voll ausgeschöpft (Borrmann et al. 2021, S. 12).

Auf der anderen Seite bedeutet Big BIM, dass alle Beteiligten in allen Disziplinen durch einheitliche Methoden und interdisziplinäre BIM-Anwendungen Modelle erhalten, mit denen sie in allen Phasen des Gebäudelebenszyklus kommunizieren können (Spengler und Peter 2020, S. 13). Ob Big BIM oder Small BIM, BIM-Teilnehmer, die für die Entwicklung von BIM-Modellen nur einen Softwarehersteller, wie z.B. Auto Desk, verwenden und die entsprechenden proprietären Schnittstellen

für den Datenaustausch nutzen, werden als Closed BIM bezeichnet, welches in Big Closed BIM und Small Closed BIM unterteilt wird (Gregor Fleischmann 2021, S. 243). Im Gegensatz dazu wird Open BIM genannt, wenn die am Bau Beteiligten BIM-Software verschiedener Hersteller verwenden, um Daten auszutauschen und Informationen über offene und neutrale Datenformate und Datenschnittstellen zu kommunizieren, und Open BIM wird auch in Big Open BIM und Little Open BIM unterteilt (Borrmann et al. 2021, S. 13).

2.3.2. Industrie Foundation Classes (IFC)

Open BIM als idealer Zustand ist jedoch nicht sehr leicht zu erreichen. Noch immer gibt es Unterschiede zwischen den Produkten verschiedener Hersteller. BuildingSMART ist es gelungen, ein herstellerunabhängiges Datenformat zu schaffen, das 2013 in einen ISO-Standard überführt wurde (Borrmann et al. 2021, S. 13–14). Dieses Datenformat enthält alle Gebäude-Informationen und -Details und wird Industrial Foundation Class (IFC) genannt. IFC bricht dieses Informationssilo auf und spielt nun eine unersetzliche Rolle in Open BIM (Blankenbach und Becker 2020, S. 789). Die IFC kann auch zur Entwicklung von Robotern verwendet werden, die mit der IFC-Datei zusammenarbeiten können, um das Potenzial der Gebäudedaten weiter auszuschöpfen und die Effizienz und Sicherheit auf der Baustelle zu verbessern.

3 Methodik

3.1. Erstellung des Verlegeplans

Für die Planung und Erstellung eines Verlegeplans für Dachdämmplatten müssen zunächst die Neigung des Flachdachs und die entsprechenden Abmessungen festgelegt werden. Sobald diese Daten vorliegen, lassen sich anhand der Neigung zunächst die Differenz (**ΔD**) zwischen dem niedrigsten und dem höchsten Punkt des Daches berechnen, wobei **n** die Neigung des Flachdachs im Entwurf darstellt (**n** > 2%) und **L** die Seitenlänge der zu berechnenden Seite ist.

$$\Delta D = n * L \qquad (1)$$

Nach der Berechnung des Höhenunterschieds zwischen dem niedrigsten und dem höchsten Punkt des Flachdachs kann der Höhenunterschied (**Δd)**, den die Dämmplatte entsprechend dieser Neigung auszugleichen hat, nach dem ähnlichen Dreiecksprinzip berechnet werden. Dabei darstellt **l** die Seitenlänge eine Dämmung.

$$\Delta d = \Delta D * l/L \qquad (2)$$

Außerdem werden alle Dachdämmplatten mit einer Höhendifferenz genormt, in der Regel 20 cm pro Stufe, so dass die berechnete Höhendifferenz einer einzelnen Dachdämmplatte auf ein ganzzahliges Vielfaches von 20 aufgerundet wird, z. B. **Δd** ist 18 cm, dann ist die Höhendifferenz einer einzelnen Dachdämmplatte 20 cm. Schließlich kann die Mindestdicke der Dämmplatten aus den Berechnungen zur Erwärmung des Gebäudes ermittelt werden, dann kann die genaue Dicke jeder einzelnen Platte berechnet werden und schließlich kann der Verlegeplan für die Dämmplatten auf der Grundlage der genauen Abmessungen der berechneten Platten erstellt werden.

Aus persönlicher Erfahrung und Beobachtung ist festzustellen, dass die Dachdecker bei der Ausführung von Flachdächern zunächst die Baumaterialien (z. B. Dämmungen, Abdichtungen etc.) auf der einen Hälfte des Daches stapeln, um die Arbeiten auf der anderen Hälfte durchzuführen. Danach werden die Baumaterialien auf dem fertigen Abschnitt gestapelt, um die restliche Hälfte des Dachs auszuführen. Für die folgenden Berechnungen kann der Grundriss des Daches als halbes Dach betrachtet werden, so dass der niedrigste Punkt des Daches zur Vereinfachung der Berechnung ganz links auf dem Grundriss angesetzt werden kann.

3.2. Umsetzung der Simulationsumgebung von BIM

BIM bietet alle Informationen über das Gebäude. Um den Roboter in einer möglichst präzisen Simulationsumgebung betreiben zu können, ist es notwendig, das 3D-BIM-Modell des Gebäudes in die Robotersimulationssoftware Gazebo zu konvertieren. Ein heutzutage häufiger Ansatz besteht darin, alle Informationen über das Gebäude direkt in eine Simulationsumgebung zu konvertieren, die über das Ifc-Plugin vom Roboter genutzt werden kann (IfcOpenShell 2022). Im Rahmen dieser Arbeit benötigt der Roboter nur die geometrischen Informationen des Gebäudes. Daher wird ein alternativer Ansatz versucht, das BIM-Modell des Gebäudes in eine Simulationsumgebung für den Roboter umzuwandeln.

3.2.1. 3D-Modelle und Lumion-Plugin in Revit

Durch den Export einer Ifc-Datei kann Revit ein Gebäudemodell für andere Zwecke verwenden. Dieses Modell kann alle Informationen über das Gebäude wie Geometrie, Materialinformationen usw. umfassen. Allerdings benötigt der Roboter für die Simulation in dieser Arbeit nur die Geometrie des Gebäudes. Daher werden andere Methoden zur Umsetzung des BIM-Modells in Revit in eine Simulationsumgebung erforscht und verwendet. Durch die Installation des Lumion-Plugins in Revit kann die 3D-Ansicht des ausgewählten Geschosses in eine Collada-Datei (.dae) umgewandelt werden (Act-3D 2022).

3.2.2. Modellanpassung

Blender ist eine Open-Source-3D-Modellierungssoftware, die nicht nur modelliert und gerendert, sondern auch über Python mit Skripten programmiert werden kann. Es unterstützt viele Formate, einschließlich Collada (.dae). Es unterstützt viele Formate, einschließlich Collada (. dae). Mit dem IfcBlender-Plug-in kann Blender Ifc-Dateien direkt importieren, wobei die importierten Modelle alle Informationen über das Gebäude enthalten (IfcOpenShell 2022).

3.2.3. Simulationsumgebung in Gazebo

Mit der Gazebo-Software können für ROS eine Simulationsumgebung entwickelt

werden. Mit der Gazebo-Software können für ROS eine Simulationsumgebung entwickelt werden. Diese Software kann zahlreiche verschiedene Geometrien hinzufügen und auch alle Eigenschaften des Objekts definieren, einschließlich der physikalischen Eigenschaften wie Gravitationsbedingungen, ob es stabil ist oder nicht, Reibungskoeffizient und Lichtquellen (Open Robotics 2022a). Es ist auch möglich, Sensoren wie LIDAR, Kameras usw. zu verwenden, indem das Paket des modularen Roboters aufgerufen wird. Dadurch wird eine äußerst realitätsnahe Simulationsumgebung für die Bewegungssimulation des Roboters geschaffen.

In Gazebo kann durch den Import einer Collada-Datei ein Modell erstellt werden. Nach dem Importieren des Modells muss es an der richtigen Position platziert und seine Stabilität als statisch ausgewählt werden. Im Anschluss wird das Modell als Gazebo-Modell in der Gazebo-Modellbibliothek gespeichert, die für den letzten Schritt der Entwicklung einer Simulationsumgebung, die Erstellung der „World", verwendet werden kann.

In Gazebo wird eine Simulationsumgebung als „World" aufgerufen und gespeichert, und die „World"-Datei hat die Dateinamenserweiterung „.world". Einer leeren Umgebung können verschiedene Elemente hinzugefügt werden. Außerdem ist es auch möglich, einige geometrische Formen selbständig einzubauen. Indem das zuvor in der Modellbibliothek gespeicherte Gebäudemodell hinzugefügt, die Höhe und Ursprungsposition angepasst und im Ordner „World" von Gazebo gespeichert wird.

3.3. Entfernungssensor

In dieser Arbeit wird ein auf einem Laserentfernungsmesser basierender SLAM-Algorithmus verwendet. Laserentfernungsmesser (LDS) sind auch als LIDAR oder LRF und Laserscanner bekannt. LDS-Sensoren werden aufgrund der Vorzüge von hoher Leistung, schneller Geschwindigkeit und Datenerfassung in Echtzeit im Bereich der Entfernungsmessung in zahlreichen Anwendungen eingesetzt. Die meisten Sensoren zur Entfernungsmessung in Robotern verwenden LDS-Lasersensoren, die zur Identifizierung von Objekten und Personen und werden aufgrund ihrer Echtzeit-Datenerfassung auch häufig in autonomen Fahrzeugen eingesetzt (Pyo et al. 2017, S. 217). Der in dieser Arbeit verwendete TurtleBot3 ist mit einem 360 Laserentfernungsmesser (LDS-01) ausgestattet. Ein typischer LDS-Sensor mit einer 360-Grad-Abtastung, der von einem Gleichstrommotor angetrieben wird. er hat eine Funkreichweite von 12 cm bis 350 cm (Robotis 2022).

Der LDS-01 verwendet Laser als Lichtquelle, um die Entfernung zwischen dem Sensor und dem zu messenden Objekt zu messen. Der LDS verwendet Laser als Lichtquelle, um die Entfernung zwischen dem Sensor und dem zu Zielobjekt zu messen. Zur Berechnung der Entfernung werden kurze Infrarot-Laserlichtimpulse von einem Laser-Entfernungsmesser ausgestrahlt und die Zeit (**Δt**) gemessen, die der reflektierte Impuls für die Rückkehr benötigt (Gurel 2018). Dabei steht **d** für die vom LDS/01 gemessene Entfernung und **c** für die Lichtgeschwindigkeit.

$$d = 1/2\ \Delta t \times c \tag{3}$$

Die Zeit von der Ausstrahlung bis zur Rückkehr des Lichts ist aufgrund der geringen Entfernungen fast unmöglich zu berechnen, so dass eine alternative Methode verwendet werden kann. Die Entfernung zwischen dem Sensor und dem Zielobjekt wird anhand der Phasendifferenz (**Δφ**) zwischen dem ausgesandten und dem zurückgesandten Laserlicht und der Frequenz des Lasers (**f**) berechnet.

$$d = 1/2\ \Delta\varphi \times 1/f \times c \tag{4}$$

Der Laserentfernungsmesser erfasst den zurückkommenden Laser, und wenn kein Licht zurückgeworfen wird, verliert der Laserentfernungsmesser seine Funktion. Daher wird die Messung ungenau, wenn ein Objekt mit einem Oberflächenmaterial mit Totalreflexionseigenschaften wie Spiegel, Klarglas usw. angetroffen wird. Außerdem kann der LDS-01 nur Objekte in der horizontalen Ebene erfassen, da sich der Lasersensor nur um 360 Grad in der horizontalen Ebene drehen kann, so dass der LDS-01 Daten in 2D misst.

3.4. SLAM Algorithmus

In diesem Abschnitt wird ein verbesserter RBPF-SLAM Google Open Source Algorithmus „Gmapping“ eingesetzt, um Indoor-Gitterkarten von zwei simulierten Umgebungen zu erstellen. „Gmapping“ ist ein effizienter Rao-Blackwellized Partikelfilter, welcher Gitterkarten anhand von Laser Entfernungsmessungen erlernt (Norzam et al. 2019, S. 5) (Liu et al. 2020, S. 1). Bei Gmapping trägt jedes Partikel eine individuelle Karte der Umgebung, und durch die Berechnung der exakten vorgeschlagenen Verteilung (proposal distribution) wird die Anzahl der Partikel reduziert, um die Bewegung des Roboters und die aktuellen Beobachtungen zu berücksichtigen, so dass die Unsicherheit in der Position des Roboters stark reduziert wird (Zhang et al. 2020, S. 2; Grisetti et al. 2007).

Das Wesentliche von SLAM ist ein probabilistisches Problem (Stachniss et al. 2016, S. 1154). Wenn sich der Roboter in einer völlig unbekannten Umgebung befindet, kann man die Zeit als **T** definieren. Die genaue Position des Roboters relativ zur Umgebung zu einem bestimmten Zeitpunkt kann durch die Werte von drei Dimensionen beschrieben werden: die Koordinaten des Roboters auf einer zweidimensionalen Karte und der Winkel des Roboters relativ zum Koordinatensystem der Karte, der durch $\mathbf{X_T}$ dargestellt werden kann. Die Position des Roboters zu verschiedenen Zeiten ist demnach:

$$X_T = \{\mathrm{x}_0, \mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_t\} \tag{5}$$

Mit Ausnahme von $\mathbf{X_0}$, das als Startkoordinate des Algorithmus bekannt ist, kann keine der anderen Koordinaten durch Abtasten ermittelt werden. Anhand der Angaben des Odometers lassen sich dann die relativen Koordinaten des neuen $\mathbf{X_T}$ bestimmen. Zu den Informationen des Odometers werden die Anzahl der Radumdrehungen und der Lenkeinschlag gezählt, die vom Raddrehgeber abgerufen werden können. Definiert $\mathbf{U_T}$ als die relative Kilometerleistung zwischen zwei aufeinanderfolgenden Zeitpunkten T-1 und T, kann $\mathbf{U_T}$ wie folgt ausgedrückt werden:

$$U_T = \{u_0, u_1, u_2, \dots, u_t\} \tag{6}$$

Der Roboter kann mit Hilfe der LIDAR-Anlage Objekte in der Umgebung (z.B. Wände) erfassen. Die Erfassung erfolgt als statische, realistische Information über die Umgebung, die durch eine Umgebungskarte m dargestellt werden kann. Die vom LIDAR gesammelten Informationen beschreiben die Position der Objekte und wenn diese als $\mathbf{Z_t}$ definiert werden, lässt sich ableiten, dass:

$$Z_t = \{z_1, z_2, z_3 \dots, z_t\} \tag{7}$$

Die Daten, die vom Roboter gemessen oder abgetastet werden können, sind $\mathbf{Z_T}$ und $\mathbf{U_T}$, und das Ziel ist es, eine realistische Karte der Umgebung m (Kartierung) und die relativen Koordinaten des Roboters $\mathbf{X_T}$ (Lokalisierung) zu erhalten, was der Zweck und die Bedeutung von SLAM ist. Es gibt zwei Modelle von SLAM, eines ist das vollständige SLAM, nämlich:

$$p(X_T, m | Z_T, U_T) \tag{8}$$

Die andere Variante, die im Vergleich zum vollständigen SLAM gleich wichtig ist, wird als Online-SLAM bezeichnet, dass nur die Position zu einem bestimmten Zeitpunkt

t bestimmt, nämlich:

$$p(x_t, m|Z_t, U_t) \tag{9}$$

Zur Lösung probabilistischer SLAM-Probleme gibt es zwei Methoden, den Kalman-Filter und die RBPF-Methode. (Gurel 2018). Der in dieser Arbeit verwendete SLAM-Algorithmus basiert auf der Optimierung der RBPF-Methode.

3.4.1. RBPF SLAM

Es gibt zwei Modelle im RBPF-basierten SLAM-Algorithmus: das Beobachtungsmodell **p(zi|xi,m)** und das Bewegungsmodell **p(x,|x_{t-1},u_{t-1})** (Liu et al. 2020, S. 2). Ausgehend von der Umgebungskarte **m** des mobilen Roboters und der Pose $\mathbf{X_t}$ kann das Beobachtungsmodell die Ungenauigkeit der Umgebung ableiten. Das Bewegungsmodell kann aus der Trajektorie $\mathbf{X_{t1}}$ des mobilen Roboters zum vorherigen Zeitpunkt und der Odometrieinformation **Ut1** eine neue Roboterposition $\mathbf{X_t}$ ermitteln, indem es die Wahrscheinlichkeitsdichte des Roboters darstellt, eine neue Position einzunehmen. Die A-posteriori-Wahrscheinlichkeit von RBPF ist:

$$\begin{aligned} &p(X_{1:t}, m \,|\, Z_{1:t}, U_{1:t}) = \\ &p(X_{1:t} \,|\, Z_{1:t}, U_{1:t}) \cdot p(m \,|\, X_{1:t}, Z_{1:t}) \end{aligned} \tag{10}$$

Die mögliche Trajektorie des Roboters wird durch die verschiedenen Partikel repräsentiert, von denen jedes eine eigene Karte der Umgebung besitzt, die dann durch Beobachtung des Modells aktualisiert wird. Da die Landmarken in dieser Karte unabhängig sind, kann jede Landmarke durch ein niedrigdimensionales EKF-Daten dargestellt werden, und die Karte wird als eine Reihe von Merkmalslandmarken dargestellt, die einer Gaußschen Verteilung gehorchen, und die Landmarken werden geschätzt, um eine Schätzung der Karte zu erhalten (Liu et al. 2020; Murphy und Russell 2001).

3.4.2. Gmapping SLAM

Der GMapping-Algorithmus ist ein laserbasierter SLAM-Algorithmus, der die SLAM-Methode des Rao-Blackwellized Partikelfilter verwendet (Grisetti et al. 2005). Dies ist

der wahrscheinlich am häufigsten verwendete SLAM-Algorithmus, dessen Implementierung auf openslam.org zu finden ist (OpenSLAM 2007). Der Algorithmus wurden ursprünglich von Murphy et al. (2001) entwickelt, und seine wesentliche Methode besteht darin, Rao-Blackwellized Partikelfilter (RBPFs) zu verwenden, um Zustandsübergangsfunktionen vorherzusagen. In der Forschung von Grisetti et al. (2005) wurden zwei wesentliche Verbesserungen vorgenommen, nämlich die Optimierung der Vorschlagsverteilung und die Einführung eines adaptiven Resampling, um den Algorithmus für praktische Anwendungen besser geeignet zu machen. Aufgrund der Verwendung von Gitterkarte wird es Gmapping genannt.

In der Regel ist die Ungenauigkeit der vom Odometer zur Verfügung gestellten Positions- und Richtungsinformationen des Roboters größer als die vom LIDAR ermittelte, und die Laserverteilung liegt näher an der tatsächlichen Zielverteilung als die des Odometers (Siehe Abbildung 3.1). Es ist daher notwendig, die vom LIDAR bereitgestellten Informationen in die vorgeschlagene Verteilung zu integrieren, so dass die vorgeschlagene Verteilung dem Ziel immer näher kommt (Zhang et al. 2020, S. 2).

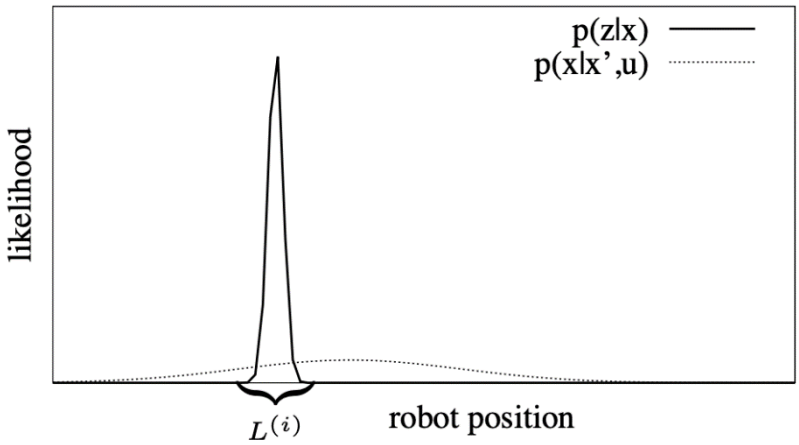

Abb. 3.1 Verteilung von LIDAR-Beobachtungsmodellen und Modellen der Bewegung des Odometers
Quelle: Grisetti et al. 2005, S. 3

Abb. 3.2 Verteilung von LIDAR-Beobachtungsmodellen und Modellen der Bewegung des Odometers
Quelle: Grisetti et al. 2005, S. 3

Die aktuellen neuesten Ergebnisse von Beobachtung werden genutzt, um die vorgeschlagene Verteilung zu verbessern:

$$p(x_t \mid m_{t-1}^{(i)}, x_{t-1}^{(i)}, z_t^{(i)}, u_t^{(i)} \cong N(\mu_t^{(i)}, \sum t^{(i)} \tag{11}$$

Aus dem Bereich mit der höchsten Wahrscheinlichkeit, der während des Scan-Matching-Prozesses gefunden wurde, wird eine Gruppe von Punkten x gezogen, um jeden der eigenen Parameter $\boldsymbol{\mu_t^{(i)}}$ und $\sum_t^{(i)}$ zu bewerten.

Das Resampling kann dazu führen, dass wertvolle Partikel aus dem Filter ausgesiebt werden, so dass der Algorithmus nur wenige Partikel enthält. Bei Gmapping mit einer verbesserten vorgeschlagenen Verteilung für die Partikelabtastung werden die Partikel jedoch weniger degradiert und müssen nicht jedes Mal neu gesampelt werden, dies wird als adaptives Resampling bezeichnet. Bei der adaptiven Resampling muss die Qualität der Partikel anhand eines gültigen Kriteriums für das Sampling-Verhältnis und der Zeit für das Resampling beurteilt werden.

$$N_{eff} = \frac{1}{\sum_{i-1}^{N}} * [(W_t^{(i)}]^2 \tag{12}$$

Wobei **N** die Anzahl der Partikel und wi das Gewicht des **i**-ten Beispiels ist. Die Verteilung wird umso schlechter geschätzt, je kleiner der Neff-Wert ist. Der Schwellenwert $\mathbf{N_{th}}$ wird auf N/2 gesetzt und Gmapping nimmt eine neue Samplingrate, wenn $\mathbf{N_{eff}} < \mathbf{N_{th}}$ ist.

3.4.3. AMCL

AMCL - Adaptive Monte Carlo Localization ist ein probabilistisches Lokalisierungssystem für mobile Roboter in zweidimensionalen Umgebungen, das als eine verbesserte Version von Monte Carlo Localization (MCL) angesehen werden kann. Das Ziel ist es, die korrekte Position des Roboters zu bestimmen, wenn er mit Hilfe der Umgebungskarte navigiert (Pyo et al. 2017, S. 358, Open Robotics 2020a).

Bei dem bereits erwähnten Partikelfilter im SLAM wird zunächst eine Zufallsverteilung aller Partikel im gesamten Zustandsraum aufrechterhalten. Während sich der Roboter bewegt, konvergieren die Partikel im Raum weiter und positionieren den Roboter schließlich an der richtigen Stelle in der Umgebungskarte.

Das Koordinatensystem Map ist das übergeordnete System des

Koordinatensystems Odom, das wiederum das übergeordnete System des Koordinatensystems base_link ist (Pyo et al. 2017, S. 329). Base_link stellt den Mittelpunkt des Roboters dar (Meeussen, W. 2010). Der Ursprung des Koordinatensystems Map stimmt mit dem Ursprung des Koordinatensystems Odom überein, das wiederum mit dem Koordinatensystem Base_link identisch ist, so dass der Anfangspunkt des Roboters das Koordinatensystem Map ist. Aufgrund der Drift des Koordinatensystems Odom wird der Fehler zwischen dem Koordinatensystem Odom und dem Koordinatensystem Map ohne LIDAR-Korrektur jedoch immer größer, so dass die Daten nicht mehr berechnet werden können. Mit dem AMCL-Algorithmus und den LIDAR-Informationen kann die Drift des Koordinatensystems Odom kontrolliert werden, was eine berechenbare Transformation zwischen den Koordinatensystemen ermöglicht.

3.4.4. Kartespeichern

Die gespeicherte Karte besteht aus zwei Dateien: einer Gitterkarte (OGM) im Format ".pgm" als portable Graustufenbildkarte, die die zweidimensionale Geometrie der Karte zeigt. Die gespeicherte Karte besteht aus zwei Dateien: einer Gitterkarte (OGM) im Format ".pgm" als portable Graustufenbildkarte, die die zweidimensionale Geometrie der Karte zeigt. In der weißen Farbe sind die freien Bereiche, in denen sich der Roboter bewegen kann, in der schwarzen Farbe die Bereiche, in denen sich der Roboter nicht bewegen kann (feste Hindernisse), und in der grauen Farbe die Bereiche, die nicht gescannt wurden und unbekannt sind. Der Graustufenwert im Bild wird als a posteriori-Wahrscheinlichkeit auf der Grundlage des Bayes-Theorems im Bereich von 0 bis 255 ermittelt und stellt die Belegungswahrscheinlichkeit (OCC) des Bereichs dar (Pyo et al. 2017, S. 325, Open Robotics 2018).

$$\mathrm{occ} = \frac{(255 - Graustufenwert)}{255} \tag{13}$$

Die Wahrscheinlichkeit der Belegung ist höher, wenn occ näher bei 1 liegt und im umgekehrten Fall ist die Wahrscheinlichkeit niedriger. Die weitere ".yaml"-Datei enthält Karteninformationen wie die Kartenauflösung (wie viele Meter pro Pixel), den Nullpunkt der Karte und den Schwellenwert, mit dem bestimmt wird, ob ein Raster belegt oder frei ist (Open Robotics 2020c).

3.4.5. Costmap

Das bei der Verwendung von SLAM erstellte Gitterkarte ermöglicht die Navigation durch den Abgleich der Roboter- und Sensorposition, der Hindernisinformationen und anhand der belegten, freien und unbekannten Bereiche der Gitterkarte (Pyo et al. 2017, S. 355). Die Costmap berechnet die Hindernisbereiche, mögliche Kollisionsbereiche und die beweglichen Bereiche des Roboters auf der Grundlage dieser vier Faktoren in der Navigation. Costmaps können in zwei Typen unterteilt werden. Eine davon ist die „global_costmap", die den Wegplan für die Navigation innerhalb eines globalen Bereichs einer festen Karte festlegt. Die andere ist die „local_Costmap“, die für die Wegplanung und Hindernisvermeidung in einem begrenzten Bereich um den Roboter herum verwendet wird (Anil Mahtani et al. 2016 S. 236). Die Größe der Local-Costmap um den Roboter herum, die von SLAM festgelegt wird, ist für den Turtlebot3 standardgemäß ein 3 mal 3 Meter großes Quadrat mit einer Auflösung von 5 cm pro Pixel. Diese Parameter können in der Konfigurationsdatei der Local-Costmap geändert werden.

Die Global-Costmap und die Local-Costmap haben denselben Zweck und werden auf dieselbe Weise berechnet. Die Kostenkarte wird durch einen Wert zwischen 0 und 255 dargestellt. Dieser Wert gibt an, ob der Roboter beweglich ist oder mit einem Hindernis kollidieren wird. Die Berechnung kann durch Konfiguration der

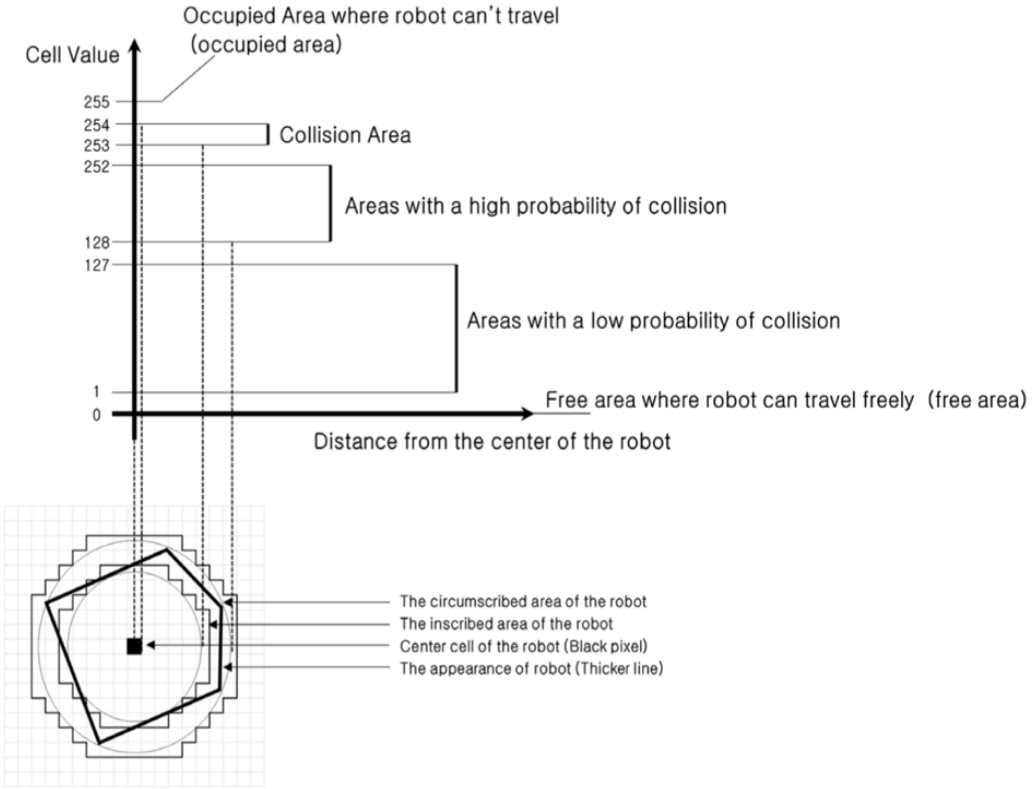

Abb. 3.3 Costmap
Quelle: Pyo et al. 2017: 356

Parameter der Costmap geändert werden. Die genaue Bedeutung der Costmap ist in Abbildung 3.2 dargestellt, wobei 0 für den Bereich steht, in dem sich der Roboter frei bewegen kann, 1-127 für den Bereich mit geringer Kollisionswahrscheinlichkeit, 128-252 für den Bereich mit hoher Kollisionswahrscheinlichkeit, 253-254 für den Bereich, der kollidieren wird, und 255 für den belegten Bereich, in dem sich der Roboter nicht bewegen kann (Pyo et al. 2017, S. 355, Open Robotics 2018).

3.5. Wegplanung

Die Wegplanung ist die Grundlage für die Navigation des Roboters, wobei das Hauptziel darin besteht, dass der Roboter den gewünschten Ort erreicht (An et al. 2021, S. 2) (Zhang et al. 2020, S. 5), und um dieses Hauptziel zu erreichen, muss der Roboter die beiden folgenden Probleme lösen:

1. Den optimalen Weg finden
2. Hindernisvermeidung während der Bewegung

Allerdings besteht aufgrund der besonderen Eigenschaften der Umgebung, in der der Roboter arbeitet (er muss mit Dachdeckern zusammenarbeiten, um Dämmung zu verlegen), und aufgrund der Genauigkeit der LIDAR-Sensoren des Roboters gibt es jedoch eine Abweichung zwischen den realen Umgebungsinformationen und den aus der Karte in SLAM erstellten Standortinformationen. Dann benötigt der Roboter zwei verschiedene Lösungsstrategien. Die globale Planung in einer statischen Umgebung ermöglicht es dem Roboter, das erste Problem zu lösen. Durch den zusätzlichen Faktor der dynamischen Veränderung der Umgebung muss jedoch eine lokale Wegplanung eingeführt werden, die es dem Roboter ermöglicht, das zweite Problem zu lösen (Zhang et al. 2020, S. 5).

Das bedeutet, dass der Roboter eine globale und eine lokale Wegplanung benötigt, um im Navigationsprozess (autonomes Fahren) zusammenzuarbeiten, damit der Roboter das grundlegende Ziel der Navigationsfunktion erfüllt, d.h. die gewünschte Position zu erzielen und dabei den optimalen Weg zu finden sowie die implementierten Hindernisse zu vermeiden zudem somit eine präzise Navigation zu erreichen. In dieser Arbeit verwendet der Roboter das Move_Base-Paket von ROS. Die Funktionsweise des Move_Base Pakets und die Algorithmen zur globalen und lokalen Wegplanung werden in den folgenden Abschnitten kurz beschrieben (Open Robotics 2020b).

3.5.1. Move_base

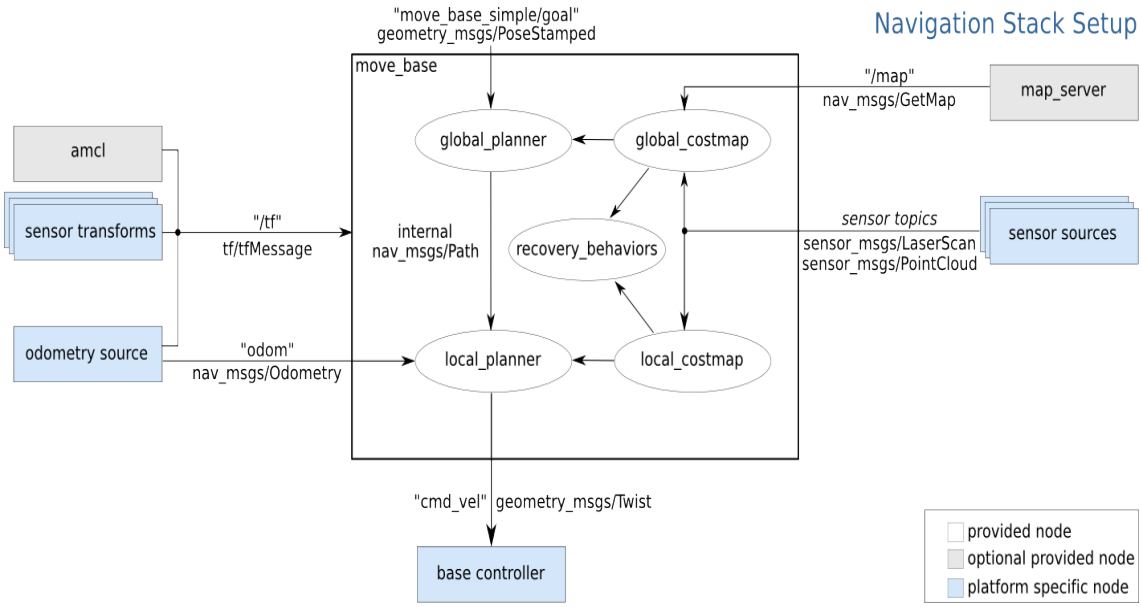

Abb. 3.4 Funktionsweise von Move_Base

Quelle: Open Robotics 2020b

Mit dem Move_Base-Paket ist es nicht mehr notwendig, die Roboterbewegung direkt durch die Steuerung der Roboterantriebsmotoren und die Verwendung des Basiscontrollers mit Odometer- und Transformationsinformationen zu realisieren (Open Robotics 2013). Dies liegt daran, dass im Move_Base-Paket nur die Koordinaten in der Karte angegeben werden müssen und der Roboter der Wegplanung zum geplanten Zielpunkt folgen kann.

Wie in der Abbildung 3.3 zeigt die detaillierte Funktionsweise des Pakets Move_Base. Innerhalb des Paketes ist es in zwei Hauptteile unterteilt, der globale Wegplaner und der lokale Echtzeit-Wegplanr. Der globale Wegplaner abonniert die Nachrichten von Sensoren und Map durch globale Costmap. Der lokale Wegplaner abonniert die Nachrichten auch von Sensoren durch lokale Costmap außerdem abonniert die lokale Wegplanung die Nachricht von Odom. Gleichzeitig veröffentlicht ein Thema zur Steuerung der Motordrehung.

Gemäß Wim Meeussen 2010 ist der Odom-Frame zwer ein fixed-world-Frame, aber aufgrund seiner häufigen Drift nicht für den langfristigen Einsatz geeignet. Der Move_base-Knoten steuert also die Drift des Odoms, indem er den AMCL-Algorithmus verwendet und das Transformen der Sensorinformationen vornimmt. Da der lokale Wegplaner, der die Bewegung des Roboters direkt steuert, auf die Odom-Nachricht abonniert ist, wird der Roboter nicht zu den exakten Koordinaten

fahren, wenn das Odom driftet (Meeussen, W. 2010).

Aus diesem Grund ist es bei der Verwendung des Knotens Move_Base wichtig, diesen im Map-Frame zu definieren. Damit soll sichergestellt werden, dass die dem Roboter eingegebenen Koordinaten stabil sind.

3.5.2. Globalpath

Der globale Planungsalgorithmus von Move_Base basiert auf navfn-Knoten und sein Algorithmus kann entweder mit Dijkstra oder mit dem A*-Algorithmus verwendet werden. In dieser Arbeit wird eine kurze Beschreibung des A*-Algorithmus gegeben (Open Robotics 2012, Open Robotics 2014). Der A*-Algorithmus ist eine globale Wegplanung, die auf der Gittermethode basiert, und seine wichtigste Grundlage ist die Kostenfunktion, die den Roboter veranlasst, einen Weg zum Endpunkt in einer bestimmten Richtung zu finden, und seine Kernfunktion ist:

$$f(n) = g(n) + h(n) \tag{14}$$

Dabei kann **n** als der nächste Zielpunkt interpretiert werden, f(n) stellt die gesamte geschätzte Funktion des aktuellen Knotens n dar, g(n) stellt die tatsächlichen Kosten vom Startpunkt bis zum aktuellen Punkt und h(n) die geschätzten Kosten vom aktuellen Knotenpunkt bis zum Endpunkt darstellen. In der Regel verwendet h(n) den euklidischen oder Manhattan-Abstand zwischen zwei Punkten im Raum, und der Wert von h(n) hat Auswirkungen auf die Leistung des Algorithmus. Die Formel zur Berechnung des Manhattan-Abstands zwischen zwei Punkten lautet wie folgt (Zhang et al. 2020, S. 5):

$$D_m = |X_1 - X_2| + |Y_1 - Y_2| \tag{15}$$

3.5.3. Localpath

Der lokale Planungsalgorithmus ist ein dynamischer Fensteralgorithmus (DWA) (Pyo et al. 2017, S. 359). Bei der globalen Wegplanung wird ein Gesamtplan erstellt, mit dem der Roboter seine Zielposition erreicht. Wie in obere Abbildung 3.3 zu sehen ist, ist die lokale Wegplanung für die Ausgabe von Geschwindigkeitsbefehlen an die mobile Einheit verantwortlich, um den Roboter sicher zum Ziel zu bewegen. Diese

Faktoren werden mit Hilfe einer Kostenfunktion, die sowohl die Entfernung zum Hindernis als auch die Entfernung zum Weg berücksichtigt, in den vom Planer erstellten Plan integriert.

Der DWA-Algorithmus erfordert eine numerische Simulation der Roboterwege über eine bestimmte Geschwindigkeitsstufe. Es ist daher notwendig, eine Darstellung des Modellzustands des Roboters zu erhalten. Ein zweirädriger Roboter mit Differentialantrieb hat keine Geschwindigkeit in Richtung der Y-Achse. Da sich der Roboter in jedem Samplingzeitraum der Programmausführung auf der Millisekundenebene befindet, kann die Trajektorie der Roboterbewegung in zwei benachbarten Samplingzeiträumen als eine gerade Linie angenähert werden. In einem Zeitraum Δ legt der Roboter eine kleine Strecke mit der Geschwindigkeit **v** unter einem Winkel **θt** zur x-Achse fest, und die Bewegungsschritte **Δx** und **Δy** des Roboters in der x- bzw. y-Achse lassen sich dann wie folgt berechnen (Zhang et al. 2020, S. 5):

$$\Delta x = x + v\Delta t cos(\theta_t) \tag{16}$$

$$\Delta y = y + v\Delta t sin(\theta_t) \tag{17}$$

Die Trajektorie des Roboters ist wie folgt gegeben:

$$X_{t+1} = x\_t + v\Delta t cos(\theta\_t) \tag{18}$$

$$Y\_(t+1) = y\_t + v\Delta t sin(\theta\_t) \tag{19}$$

$$\theta\_(t+1) = \theta\_t + w\Delta t \tag{20}$$

Hierbei ist **ω** die Winkelgeschwindigkeit des Roboters. Nachdem der Roboter-geschwindigkeit bestimmte Grenzen gesetzt wurden, wird eine Bewertungsfunktion verwendet, um die ausgewählte Trajektorie auf der Grundlage der Trajektorien zu messen, die diese Geschwindigkeitsanforderungen erfüllen, mit dem Ziel, die optimale Trajektorie auszuwählen. Diese Bewertungsfunktion lautet wie folgt (Zhang et al. 2020, S. 6):

$$G(v,w) = \sigma(\alpha head(v,w)) + \beta dis(v,w) + \gamma vel(v,w) \tag{21}$$

Dabei steht **head(v, ω)** für die geschätzte Winkeldifferenz zwischen dem Ende der Strecke und dem Ziel; **dis(v, ω)** ist der minimale Abstand zwischen dem Hindernis und der geplanten Trajektorie, **vel(v, ω)** stellt die momentane Geschwindigkeitsbewertung dar, und σ(-) ist eine Glättungsfunktion; **α, β, c > 0** sind die Bewertungskoeffizienten (Zhang et al. 2020, S. 6).

3.6. ROS Server und Client von Move_Base

Wenn SLAM und Navigation auf ein reales Projekt angewendet werden sollen, ist das oben erwähnte Move_Base-Paket erforderlich. Die im vorherigen Abschnitt durchgeführte Analyse zeigt, dass Move_Base die Koordinaten des Zielpunkts erhält, bevor es den Roboter mit Hilfe von Funktionen und Algorithmen an den angegebenen Ort fahren kann. Daher muss eine Kommunikation mit Move_Base aufgebaut werden.

Es gibt drei Hauptkommunikationsarten zwischen ROS-Knoten: Topics, Services und Actions. Bei diesem Projekt sind der Logistikroboter und der Roboterarm gemeinsam im Einsatz, und es gibt auch nicht nur eine Aufgabe für den Logistikroboter. Daher ist es notwendig, den Knoten ein Feedback zu senden, nachdem sie ihre Aufgaben erledigt haben, aber ROS Topics können kein Feedback senden und ROS Services können ein Feedback senden, aber nur einmal. Deshalb ist für diesen Fall die Kommunikation mit dem Move_Base-Knoten über Aktionen am besten geeignet.

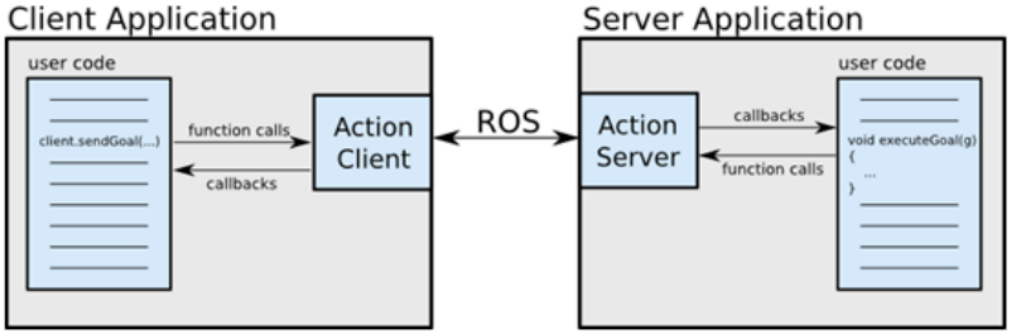

Abb. 3.5 Die Arbeitsweise von ROS Action
Quelle: Open Robotics 2018b

ROS Action scheint auch ein Frage- und Antwort-Kommunikationsmechanismus zu sein, ähnlich wie der ROS-Dienst. Zum Gegensatz zum ROS-Service verfügen Aktionen auch über einen Echtzeit-Feedback-Mechanismus, der eine kontinuierliche Rückmeldung über den Fortschritt der Aufgabe liefert und während der Ausführung

der Aufgabe beendet werden kann.

Wie die Abbildungen 3.4 und 3.5 zeigen, basieren ROS-Actions auch auf einem Server/Client-Modus, bei dem der Server und der Client über das von der Actionlib definierte Action-Protocol miteinander kommunizieren können, wobei es sich um ein Kommunikationsprotokoll handelt, das dem Benutzer eine Schnittstelle zwischen dem Server und dem Client bietet, wie in der Abbildung dargestellt. Wobei "goal" für das Ziel der Aufgabe steht, "cancel" für die Aufforderung, die Aufgabe abzubrechen, "status" für die Mitteilung an den Client über den aktuellen Status, "feedback" für eine regelmäßige Rückmeldung über die Überwachungsdaten des Vorgangs, "result“: das Ergebnis der Aufgabe wird an den Client gesendet, dieses Topic wird nur einmal veröffentlicht (Open Robotics 2019c).

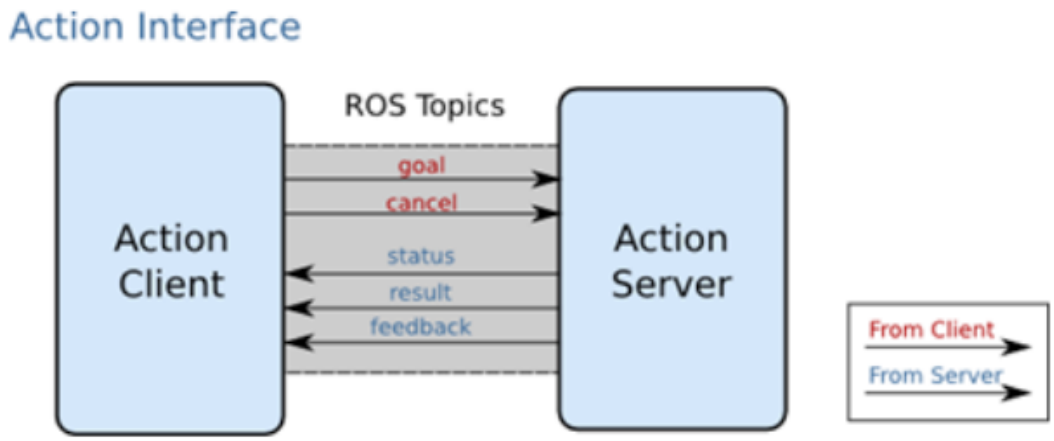

Abb. 3.5 Kommunikationsweise von Ros Action
Quelle: Open Robotics 2019c

3.7. Fernbedienung des Roboters

Das Betriebssystem von Turlebot3 ist auf einem Raspberry Pi installiert. Als kleines eingebettetes System ist es zweifellos nicht so leistungsfähig wie ein Computer, und als mobiler Roboter ist es sehr schwierig, den Roboter durch seinen Bildschirm oder Tastatur zu bedienen, während er in Betrieb ist. Daher ist die beste Lösung, das Robotersystem mit einem leistungsfähigen Computer zu verbinden, auf dem mehrere Softwarepakete und Algorithmen laufen, während das Robotersystem lediglich Informationen von den Sensoren aufnimmt, dann an den Computer weiterleitet und von diesem die Befehle zur Steuerung der Motorbewegungen des Roboters erhält.

ROS ist auf verteiltes Computing ausgelegt, d.h. der Informationsaustausch zwischen den Nodes ist möglich, wenn sich der ROS Node im gleichen Netz des Master Nodes befindet und mit dem Master Node verbunden ist (Open Robotics 2019a). Die Rahmenbedienungen dafür sind:

- Der mit der Hardware kommunizierende Treiberknoten muss auf dem mit der Hardware verbundenen Gerät laufen, zum Beispiel dem Raspberry Pi auf dem Roboter.
- Alle Nodes sind über ROS_Master_URI mit demselben Master-Node verbunden.
- Die Verbindung zwischen allen Geräten muss in beide Richtungen und über alle Ports erfolgen.
- Muss auf jedem Gerät mit einem Namen benannt werden, der von anderen Geräten aufgelöst werden kann.

Das Robotersystem verfügt auch nicht über einen Bildschirm, so dass es möglich ist, alle diese Operationen über SSH auf den Computer zu übertragen. Der Roboter und der Computer werden zunächst mit demselben WLAN verbunden, wodurch der Roboter und der Computer über SSH und die IP-Adresse des Roboters verbunden werden können. Anschließend wird im SSH-Fenster die Bringup-Lauchdatei des Roboters remote gestartet, wodurch alle sensorrelevanten Nodes auf einmal gestartet werden. Danach ist es möglich, einen beliebigen ROS-Node auf dem Computer zu starten und der Roboter gibt das entsprechende Feedback .

3.8. Eigenepacket in ROS

In den folgenden Simulationen wird es notwendig sein, die erforderlichen Programme in ROS selbst zu programmieren, um den Roboter die entsprechenden Aufgaben ausführen zu lassen. In diesem Abschnitt werden zunächst der ROS Workspace und das Programmpaket erstellt, mit dem die Programmierung durchgeführt werden kann. Ein Catkin-Workspace enthält in der Regel einen Source-, einen Build- und einen Development-Space. Pakete im workspace werden im Source Space platziert, so dass mehrere voneinander abhängige Pakete auf einmal erstellt werden können (Open Robotics 2017).

Ein ordnungsgemäßes Catkin-Package muss die folgenden drei Voraussetzungen erfüllen:

- Es muss eine package.xml-Datei vorliegen, die mit der catkin-Spezifikation übereinstimmt.
- Muss eine CMakeLists.txt-Datei in der Catkin-Version besitzen.
- Muss sein eigenes Verzeichnis aufweisen.

3.9. Canny operator in OpenCV

In diesem Artikel ist OpenCV jedoch unabhängig vom ROS-System, d. h. OpenCV läuft nicht als ein Knoten, obwohl es am Betrieb des Roboters beteiligt ist. Aufgrund der Freundlichkeit von OpenCV in Bezug auf Maschinenlernen und der Unterstützung von ROS wurde beschlossen, OpenCV als Methode für das Computer-Vision-Verfahren zu wählen und bei zukünftigen Forschungen mit ROS zu arbeiten.

Die Erkennung des Verlegezustandes der Dämmplatten und die Identifizierung der Platten erfolgt mit Hilfe von Computer Vision. Wie bereits erwähnt, wird das gleiche Dämmmaterial auf dem Flachdach verlegt.

Die Identifizierung und Kategorisierung von Dämmplatten basiert auf dem Dicken der Dämmplatten. Da die Dämmplatten mit einem Gefälle verlegt werden und die Dicke der Platten je nach Verlegeplan unterschiedlich ist, müssen die Dämmplatten ein ebenes Gefälle bilden, damit das Regenwasser vom Dach abfließen kann. Die verschiedenen Dämmplatten sind nur in bestimmten Bereichen verlegbar, weshalb sie vor der Verlegung identifiziert werden müssen.

Da die Seitenfläche der Dämmplatte eine rechteckige Form hat, können zum Messen der Dicke der Seitenfläche der Dämmplatte direkt die Länge des Rechtecks gemessen werden, um die Höhe der Seitenfläche der Dämmplatte zu erhalten, die auch die Dicke der Dämmplatte ist. Um den Messfehler zu kontrollieren, werden die beiden Seitenlängen des Seitenrechtecks und der Abstand zwischen den Mittelpunkten getrennt gemessen, und der Durchschnitt der drei Längen entspricht der Dicke der Seite der Dämmplatte.

Für die Verlegung von Dämmplatten gibt es keine Norm, so dass der Dachdecker die Platten nur fugenlos und straff miteinander verlegen kann. Die Methode zur Überprüfung der Verlegung der Dämmplatten ist daher dieselbe, wobei überprüft wird, ob die Dämmplatten erfolgreich verlegt wurden und keine Fugen (Kanten) zwischen den beiden Dämmplatten vorhanden sind.

Daher werden in dieser Arbeit Canny's operator, der 1986 von John Canny vorgeschlagen wurde, zur Identifizierung von Kanten verwendet, es handelt sich um einen mehrstufigen Algorithmus zur Kantenerkennung (Juan 2017, S. 1). Dazu wird hauptsächlich ein Gauß-Filter verwendet (Rong et al. 2014, S. 577) und der Intensitätsgradient des Bildes ermittelt, indem zunächst das RGB-Bild in eine

$$Gray = R * 0.229 + G * 0.587 + B * 0.114 \quad (22)$$

Graustufenkarte umgewandelt wird.

3.9.1. Gauß-Filter

Um den Canny-Operator für die Kantenerkennung zu verwenden, wird das Bild zunächst mit einem Gauß-Filter geglättet (Xu et al. 2021, S. 3). Das Bild muss daher gefiltert werden, um das Rauschen zu entfernen und einige der Nicht-Kantenbereiche des Bildes, die eine schwache Textur aufweisen, zu glätten.

Die Größe des Kernels eines Gauß-Filters wirkt sich auf die Leistung des Detektors aus. Im Allgemeinen gilt: Je größer die Detektorgröße, desto unempfindlicher ist der Detektor gegenüber Rauschen. Die Größe des Kerns wird im Allgemeinen durch eine Matrix mit ungeraden Größen bestimmt. Das folgende Beispiel zeigt einen 5x5 als Größe Gauß- Kernels(OpenCV 2022c, Bradski et al. 2008: 112).

$$B = \frac{1}{159}\begin{vmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{vmatrix} * A \quad (23)$$

3.9.2. Intensitätsgradienten des Bildes

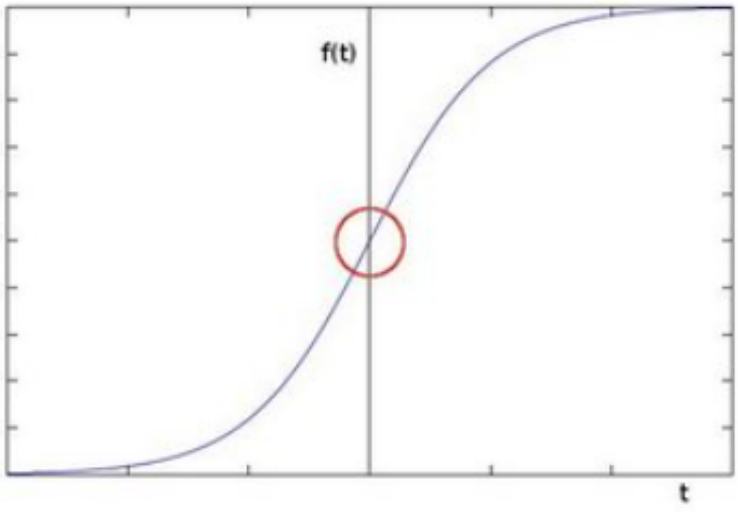

Abb. 3.6 Verteilungsfunktion von Pixeln und Intensität

Quelle: Open CV 2022a

Nach der Umwandlung des Bildes in ein Graustufenbild und der Glättung mit einem Gauß-Filter wird das geglättete Bild mit dem Sobel-Kernel sowohl in horizontaler als auch in vertikaler Richtung gefiltert (Juan 2017, S. 1),

Die Hauptlogik des Canny-Operators besteht darin, die Stelle im Bild zu finden, an der die Grauintensität am stärksten variiert. Wie in den beiden folgenden Abbildungen 3.6 und 3.7 dargestellt sind, wird angenommen, dass es ein

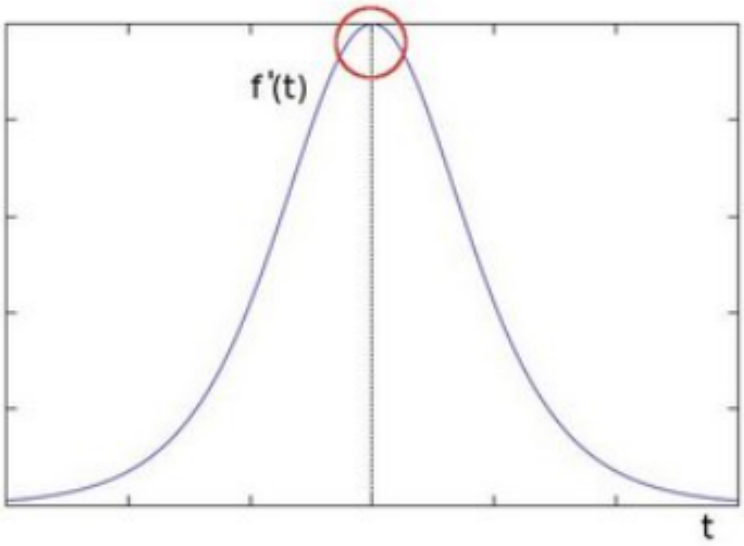

Abb. 3.7 Ableitungen erster Ordnung von Pixel-Intensitätsfunktionen

Quelle: Open CV 2022a

eindimensionales Punktbild gibt, in dem die Kanten durch Variationen der Pixelintensitäten dargestellt werden, was durch die Ableitung erster Ordnung der

Intensitäten intuitiv besser zu erkennen ist.

Der endgültige Kantengradient G kann wie folgt abgeleitet werden:

$$G = \sqrt{G_x^2 + G_y^2} \tag{24}$$

Die Richtung der Gradiente ist:

$$\theta = tan^{-1}(\frac{G_x}{G_y}) \tag{25}$$

Nachdem der Gradient und die Ausrichtung aller Pixelpunkte festgestellt wurden, müssen die Pixelpunkte im Bild durchlaufen und alle Punkte, die keine Kanten sind, entfernt werden. Dazu wird ermittelt, ob der aktuelle Pixelpunkt den Maximalwert des Gradienten unter den umliegenden Pixelpunkten mit der gleichen Gradientenrichtung aufweist. Wenn dies der Maximalwert ist, wird der Punkt beibehalten, wenn nicht, wird der Punkt unterdrückt, so dass sein Grauwert 0 beträgt (Open CV 2022b).

3.9.3. Schwellenwerte für Canny

Die Schwellenwertbildung hat schon immer eine sehr wichtige Rolle gespielt, und der Canny-Operator bietet zwei Schwellenwertoptionen (Juan 2017, S. 1). Dabei handelt es sich um den obersten und den untersten Schwellenwert. Wenn der Gradient eines Pixels höher liegt als der oberste Schwellenwert, wird es direkt als Kantenpixel bestimmt und im Bild weiß dargestellt, was als starke Begrenzung bezeichnet wird, wie Punkt A in der Abbildung 3.8. Ist der Gradient eines Pixels geringer als der niedrige Schwellenwert, wird es direkt als Nicht-Rand-Pixel eingestuft und einfach verworfen, da es im Bild als schwarz und nicht als Rand erscheint. Liegt der Gradient eines Pixels zwischen dem obersten und dem untersten Schwellenwert, wird er als schwacher Rand bezeichnet und anhand der mit ihm verbundenen Pixel bestimmt, wie Punkte B und C in der Abbildung 3.8. Wenn sein Nachbarpixel ein Kantenpixel ist (Punkt C), wird dieses Pixel ebenfalls als Kantenpixel gewertet. Wenn sein Nachbarpixel ein Nicht-Kantenpixel ist (Punkt B), das verworfen wird, dann wird dieser Pixel mit einem Gradientenwert zwischen dem obersten und dem untersten Schwellenwert ebenfalls verworfen (Bradski und Kaehler 2008, S. 152, Open CV 2022b).

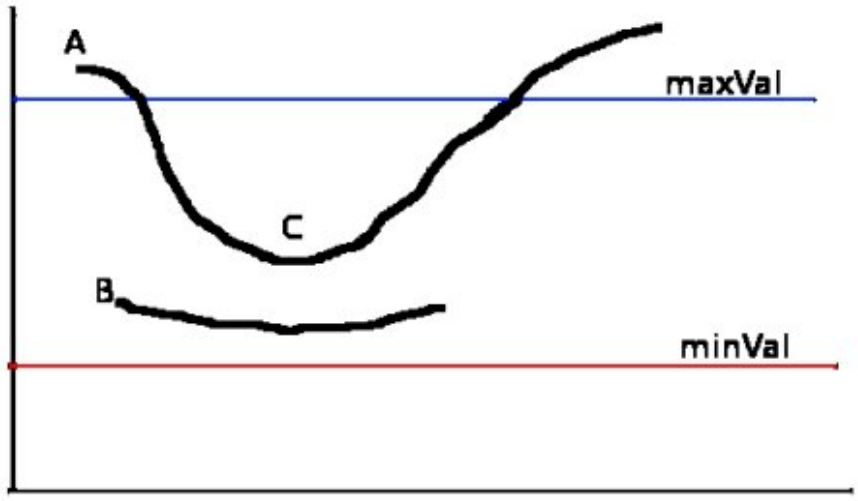

Abb. 3.8 Schwellenwerte von Canny
Quelle: Open CV 2022b

Das bedeutet, dass, wenn der oberste und der unterste Schwellenwert beide sehr hoch sind, dann werden Pixel entfernt, die sich sonst auf einer Kante im Bild befinden würden, jedoch deren Gradient die Schwellenkriterien nicht erfüllt, was zu einem Bild mit zu wenig Kanteninformation führt. Wenn der oberste und der unterste Schwellenwert zu niedrig angesetzt sind, werden Pixel, die sich nicht an der Kante befinden, aber ein Gradient aufweisen, der über dem obersten Schwellenwert liegt, als Kanten im Bild angezeigt, was zu vielen redundanten und falschen Informationen im Bild führt. Wie in der Abbildung.

Da die verschiedenen Materialien unterschiedliche Texturen und Farben der Dämmplatten aufweisen, gelten auch unterschiedliche oberste und unterste Schwellenwerte für die jeweiligen Materialien. Bei der Implementierung der nachfolgenden Beispiele wird es notwendig sein, experimentelle Untersuchungen durchzuführen, um die obersten und untersten Schwellenwerte für jedes der verschiedenen Materialien zu ermitteln.

4 Arbeitsablauf mit Simulationen

Zur Verifizierung des obigen Ansatzes werden zwei Simulationen durchgeführt. Die erste besteht darin, eine Simulationsumgebung für das Dach zu entwickeln, indem zunächst ein Dachverlegeplan entworfen wird, um die Zielpunkte zu bestimmen, die der Roboter während der Simulation erreichen soll, und dann ein Programm zu schreiben, das den Roboter nacheinander zu den angegebenen Koordinaten fahren lässt. Für die Computervision wurden Experimente durchgeführt, um die Schwellenwerte für verschiedene Dämmplattenmaterialien mit dem Canny-Operator zu bestimmen. Letztendlich wird diese Simulation zeigen, dass der Roboter die Aufgabe eines Logistikroboters mit Hilfe von ROS unter Verwendung von SLAM, Wegplanung und dem Canny-Operator erfüllen kann.

Die zweite ist die Kombination des Instituts für Bauinformatik mit seiner Simulationsumgebung. In der Baupraxis gibt es oft temporäre Hindernisse (z. B. Baumaschinen, Baumaterialien, Bauarbeiter usw.), die nicht Bestandteil des Entwurfs und des Werkplans sind, was das SLAM durch den Roboter auf der Baustelle ineffizienter macht. Darüber hinaus kann der SLAM-Prozess des Roboters auch die anderen Aufgaben der Bauarbeiter auf der Baustelle beeinträchtigen. Außerdem sind temporäre Hindernisse nicht ortsfest, was dazu führen kann, dass der Roboter durch SLAM eine Karte erstellt, die nicht mit der tatsächlichen Situation übereinstimmt, was wiederum den Roboter daran hindert, seine Arbeit auf der Baustelle fortzusetzen. Dazu ist am besten eine Karte in einer Simulationsumgebung durch SLAM zu erstellen und dann die zuvor erstellte Karte in der Simulationsumgebung für die Roboternavigation, die Hindernisvermeidung und die Wegplanung auf der realen Baustelle zu verwenden.

Der Arbeitsablauf der Simulation besteht daher darin, zunächst eine völlig gleiche Simulationsumgebung auf der Grundlage des BIM-Modells des Instituts zu erstellen. Der Roboter nutzt SLAM, um eine Karte in der Simulationsumgebung zu erstellen. Der Roboter wird dann in der realen Umgebung eingesetzt und nutzt die in der Simulationsumgebung erstellte Karte zur Navigation und zur Erkennung fester Hindernisse während des Navigationsprozesses sowie zur Entdeckung temporärer Hindernisse und zur automatischen Hindernisvermeidung mittels LIDAR während des Fahrprozesses.

Der Turtlebot3 WafflePi wird als Logistikroboter eingesetzt, sowohl im praktischen Einsatz als auch in Simulationsumgebung. Hauptaufgaben werden SLAM und autonomes Fahren sein.

Offensichtlich ist, dass der Roboter aufgrund seiner geringen Größe und Reifen über einen sehr geringen Abstand zum Boden verfügt. Der Turtlebot3 ist daher für den Einsatz und das Testen auf einer realen Baustelle nicht geeignet. Deshalb wird der Einsatz des Roboters innerhalb des Instituts für Bauinformatik und in einer Simulationsumgebung stattfinden, für die der Turtlebot3 ausreichend ist. Das von dem Unternehmen Robotis zur Verfügung gestellte Open-Source-Softwarepaket für Turtlebot3 wird auch intensiv genutzt.

4.1. Rahmenbedingungen

Zuvor mussten geeignete Rahmenbedingungen für die Implementierung festgelegt werden.

- Aufgrund des begrenzten Radius der Räder des Roboters. Obwohl es sich bei den Hintergrundbedingungen um eine Simulation einer realen Baustelle handelt, wird davon ausgegangen, dass der Boden vollkommen eben ist und es keine Hindernisse wie Bretter, Wasserrohre usw. gibt, die die Bewegung des Roboters behindern würden.
- Die Simulationsumgebung für das Institut und das Dach wurde anhand des BIM-Modells entwickelt, das nur die Geometrie und keine Texturinformationen enthält.
- Die Türe im Gebäude wurde in Blender beim Anpassen des 3D-Modells entfernt und der Roboter kann den Raum frei betreten
- Aufgrund einer defekten 3D-Kamera im Projekt konnte der Roboter nur die zuvor im Computer gespeicherten vorgefertigten Bilder lesen, weshalb OpenCV nicht als ROS-Knoten gestartet wurde.
- Die Simulationsumgebung des Daches von der ersten Simulation ist zu leer, es enthält außer den Außenwänden keine weiteren Gebäudeelemente, was den Roboter aufgrund seiner Größe und des eingesetzten LIDARs ebenfalls überfordert. Dies würde dazu führen, dass die Erstellung von Karten bei SLAM sehr schwierig ist und die erstellten Karten würden nicht mit der Simulationsumgebung übereinstimmen (siehe Abbildung 4.1). In diesem Fall wird die Größe des Daches in Blender angemessen verkleinert und die

Simulationsumgebung wird neu erstellt, wie in Abschnitt 4.2.3. beschrieben.

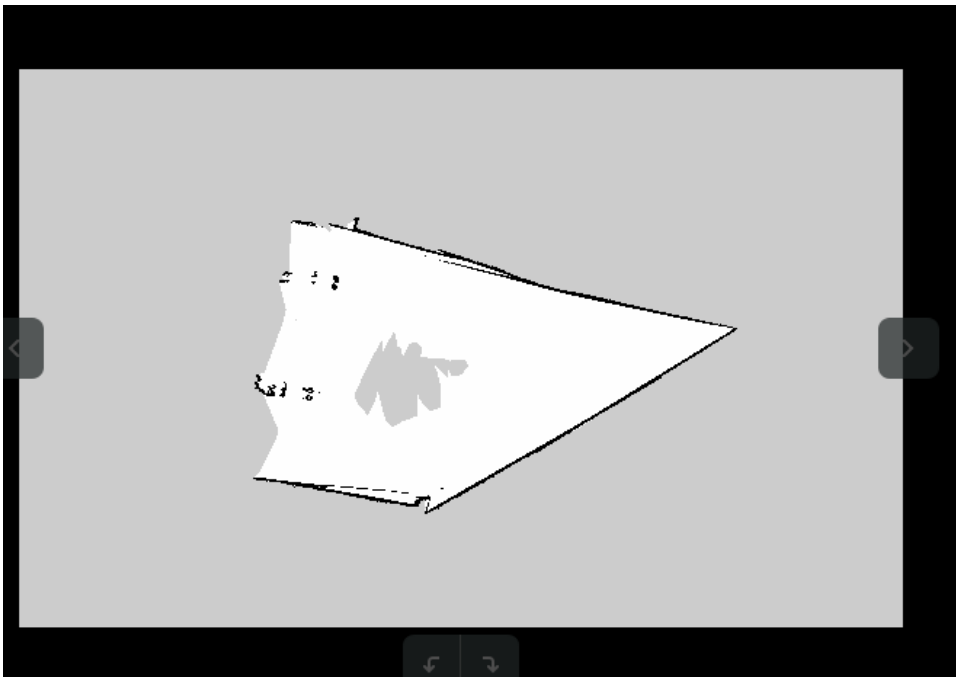

Abb. 4.1 Fehlerhafte SLAM-Karte
Quelle: Screenshot in Ubuntu

4.2. Flachdach-Simulation

In diesem Abschnitt wird der Roboter als Logistikroboter auf dem Flachdach eingesetzt, indem ein Python-Script entwickelt wird. Die Hauptaufgaben sind:

- Unterstützung des Roboterarms bei der Verlegung der Dachdämmplatten auf dem Flachdach in einer vorgegebenen Reihenfolge.
- Am Baumaterial wird die Dicke der Dämmung mit Hilfe von Computer Vision identifiziert, um die nächstfolgende Dämmplatte zu bestimmen.
- Nach der Verlegung der Dämmplatten durch den Roboterarm prüft der Roboter mit Hilfe von Computer Vision, ob die Dämmplatten ordnungsgemäß verlegt wurden.

4.2.1. Arbeitsablauf des Roboters

Aufgrund der speziellen Bedingungen des Einsatzortes muss der Roboter mit einem Kran auf die Dachflächen transportiert werden. Der niedrigste Punkt im Verlegeplan wird als der Punkt definiert, an dem der Roboter nach oben transportiert wird, und ist der Nullpunkt in der Simulationsumgebung. Nachdem der Roboter transportiert wurde, fängt er mit seiner Arbeit an. Der konkrete Arbeitsablauf sieht dabei wie folgt aus:

1. Wenn der Roboter in das Navigationspaket geladen wird, wird er mit AMCL positioniert, um den Ursprung des Koordinatensystems von Odom mit dem Koordinatensystem der Karte auszurichten.
2. Während der Roboter bereits am Nullpunkt ist, wendet er sich direkt dem Baumaterial zu, um ein Foto zu machen, anhand dessen die benötigten Dachdämmplatten identifiziert werden können.
3. Nach der Identifizierung folgt man dem Verlegeplan bis zur Position der ersten zu verlegenden Deckenplatte, stellt die Position und Richtung ein und richtet die zu verlegende Fläche aus.
4. Nachdem der Roboterarm seine Arbeit erledigt hat, macht die Kamera ein Foto und erkennt der Roboter anhand des Bildes, ob die Dämmplatten erfolgreich verlegt wurden.
5. Anschließend, falls noch nicht alle Dämmplatten der gleichen Kategorie verlegt wurden (gemäß Verlegeplan), fährt der Roboter zur nächstmöglichen Position, um die Dämmung zu verlegen, und wiederholt den vorherigen Schritt. Falls alle Dämmungen der gleichen Kategorie verlegt wurden, kehrt der Roboter zum Nullpunkt zurück und wiederholt Schritt 1, indem er die nächste Kategorie von Dämmungen identifiziert, aufnimmt und an die entsprechende Position verlegt.
6. Zum Schluss beendet der Roboter seine Arbeit, bis alle zu verlegenden Dämmungen verlegt sind.

4.2.2. Dach des Projekts

Bei dieser Arbeit handelt es sich um ein Simulationsprojekt. Für die Simulation der Verlegung von Dämmplatten für ein Flachdach wird eine Musterbaustelle ausgewählt. Das ausgewählte Dach befindet sich auf dem niedrigeren Dach des

Gebäudes an der Nürnbergstraße 31 a und der Grundriss ist in Abbildung 4.2 dargestellt.

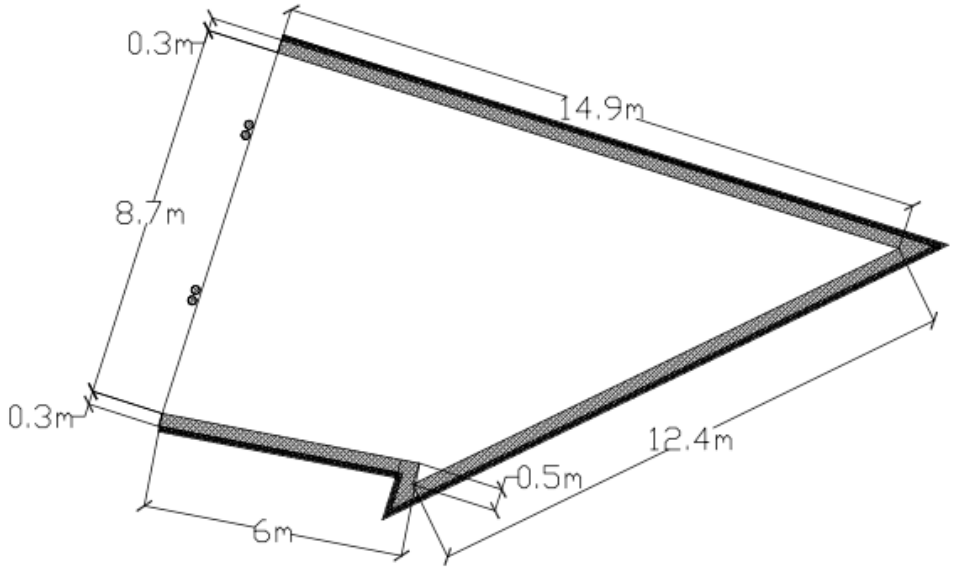

Abb. 4.2 Grundriss des Dachs
Quelle: Eigenes Zeichnen

4.2.3. Gefälle und Verlegeplan

Dämmungen, die aus EPS, PU, und Mineralwolle machen, können als Gefälle-dämmungen für Warmdächer verwendet werden[4]. Nach den technischen Angaben auf der Webseite[5], die von Dittrich GmbH mitgeteilt wird, sind die alle Dämmung in den Größen 2000mm * 1200mm mit einer Mindestdicke von 60 mm ausgestattet. Zwischen den verschiedenen Dickenstufen beträgt der Unterschied 20 mm. Um die Kalkulation zu vereinfachen, wird die Form des Daches zu einem rechtwinkligen

[4] Vgl. Kurze Meeting mit J. Dittrich von Dachdeckermeister Claus Dittrich GmbH, am 18.07.2022

[5] Vgl. https://www.isobouw.de/produkte/flachdach/flachdach-daemmplatte.html; https://www.bauder.de/de/flachdach/flachdach-produkte/waermedaemmung/flachdach/bauderpir-fa.html; https://www.rockwool.com/de/produkte/produktuebersicht/bitrock/;

Trapez mit zwei zueinander parallelen Seiten und den Längen der drei Seiten 15m * 8,7m * 6m modifiziert. Die Längen der beiden Seiten der Dämmplatten werden ebenfalls zur Erleichterung der Berechnung auf 2000mm * 1740mm festgelegt. Die Dicke der Dämmplatten am niedrigsten Punkt des Daches beträgt 60mm, wobei die Dicke der Platten an dem niedrigsten Punkt des Daches 60 mm beträgt. Bei einer Neigung von 2z % beträgt der Höhen-unterschied einer Platten auf der langen Seite 2000 mm * 2 % = 40 mm. Damit beträgt die Neigung an der kurzen Seite der Dämmplatte 40 mm/1740 mm = 2,30 % > 3 %, was der Mindestgefälle für Flachdächer entspricht.

Der Verlegeplan ist in Abbildung 4.3 dargestellt. Die in der Abbildung gezeigte Zahl 60 steht für die Dicke von 60 mm an dem niedrigsten Punkt des Dachs. Weitere Zahlen, wie z. B. 6-10, stehen für die Mindestdicke von 6 cm (60 mm) und die maximale Dicke von 10 cm (100 mm) für die Gefälledämmung. Gemäß den Vorgaben des Projekts werden die Dämmplatten am niedrigsten Punkt des Dachs, die Platten mit einer Kehle (K1-K2) und die nicht rechteckigen Platten (K3-K6, L1-L5) manuell von Dachdecker verlegt und der Roboter im Projekt verlegt den Rest der Dämmungen.

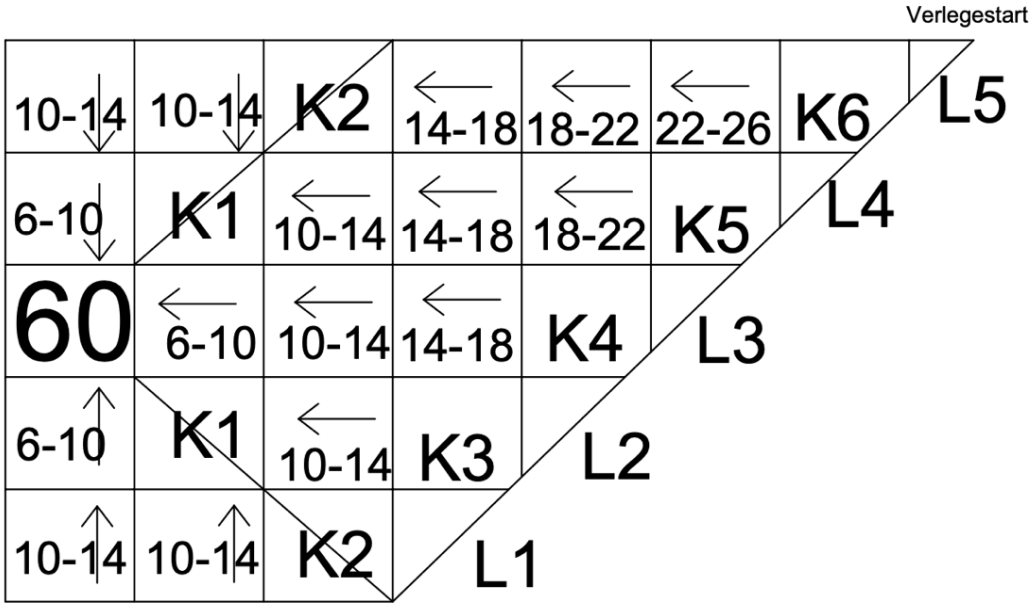

Abb. 4.3 Verlegeplan der Dämmungen
Quelle: Eigenes Zeichnen

Dem Verlegeplan zufolge sollten die Dachdecker bei L5 mit dem Verlegen starten, und sobald L5-L1 und K6-K3 verlegt sind, wird der Roboter zu verlegen anfangen. Abschließend verlegen die Dachdecker die Dämmungen K2, K1 und die am niedrigsten Punkt. Der Roboter muss zunächst zu der Stelle fahren, an der das Baumaterial platziert ist, um die einzelne Kategorie von Dämmplatten (gleiche Dicke)

aufzunehmen. Die Stelle, in dem sich der niedrigste Punkt des Daches befindet, gilt als die Stelle, an der die Baumaterialien (Dämmungen etc.) platziert werden. Um die

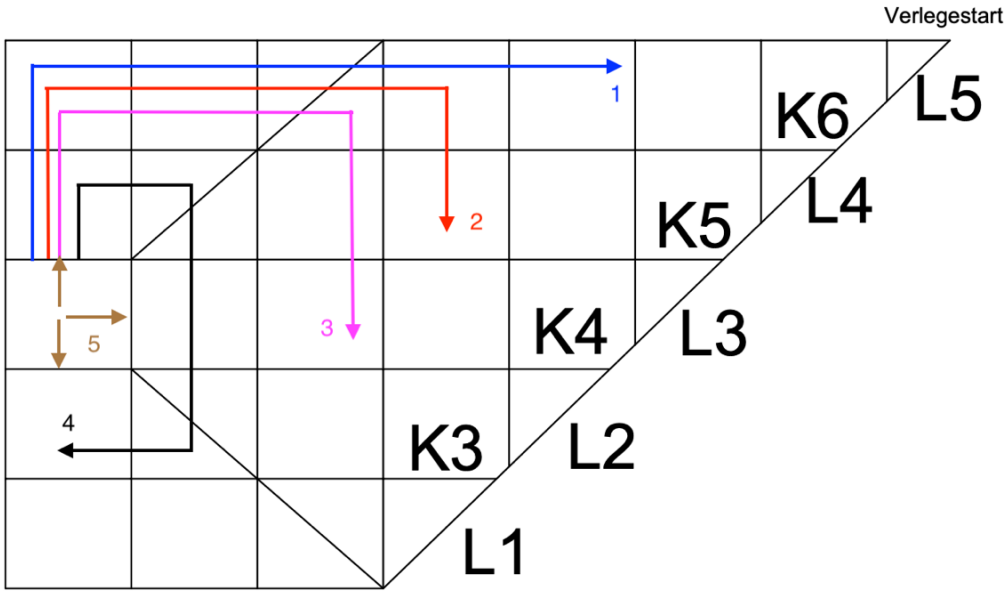

Abb. 4.4 Reihenfolge der Verlege der Dämmungen durch Roboter
Quelle: Eigenes Zeichnen

Effizienz des Roboters beim Verlegen des Daches zu erhöhen, wird er die Reihenfolge der Verlegung einhalten, wie in der folgenden Abbildung 4.4 dargestellt.

4.2.4. Entwicklung der Simulationsumgebungen

In Abbildung 4.5 ist das Dach in Rot eingekreist. Der ausgewählte Bereich wurde mit dem Lumion-Plugin exportiert und das exportierte Dachmodell wird weiterhin in Blender geöffnet. Wie in Abschnitt 4.1 bereits als Problem angesprochen, ist die ursprüngliche Größe des Daches zu groß für den Turtlebot und das LIDAR LDS-01. Der SLAM von Turtlebot auf dem Flachdach ist bei dieser Größe unmöglich. Das Dach wird in Blender in der Größe verändert.

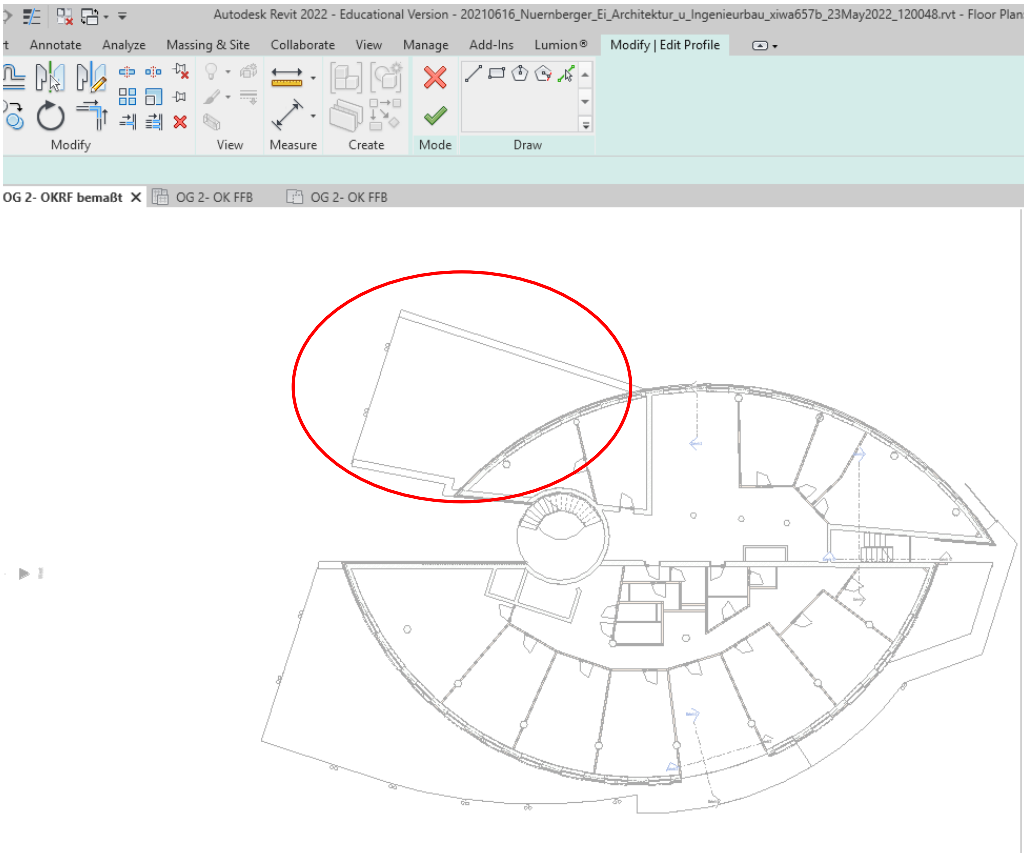

Abb. 4.5 Dach in Revit (Screen Shot)

Quelle: BIM-Modell von Fakultät Bauingenieurwesen an der TU Dresden

Die Größe des verkleinerten Flachdachs ergibt sich aus Messungen und beträgt 8,8 m an der längsten Seite. Die Maße der Dämmplatte betragen 1,22m*1m, gemessen nach dem gleichen Verhältnis. Anhand der CAD-Zeichnung des Flachdachs lassen sich die Positionen und der Winkel berechnen, in dem der Roboter beim Verlegen der einzelnen Dämmplatten fahren muss.

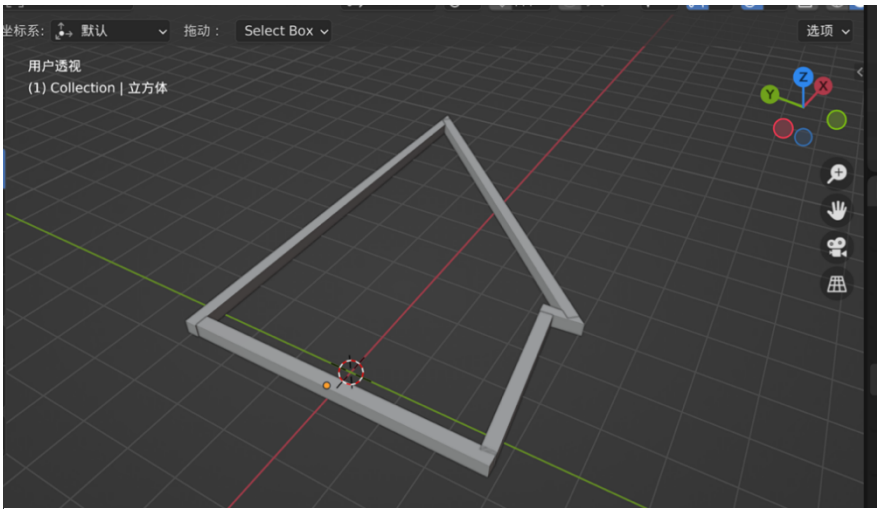

Abb. 4.6 Dachmodell in Blender verkleinert

Quelle: Eigene Modellierung

Schließlich wird das verkleinerte 3D-Modell aus Blender in Gazebo importiert, um die Entwicklung der ersten Simulationsumgebung abzuschließen. Da der vereinfachte Verlegeplan ein rechtwinkliges Trapez ist, liegen die drei Seiten des Trapezes orthogonal zum Gazebo-Koordinatensystem, d. h. in derselben Richtung wie die ursprüngliche Ausrichtung des Roboters

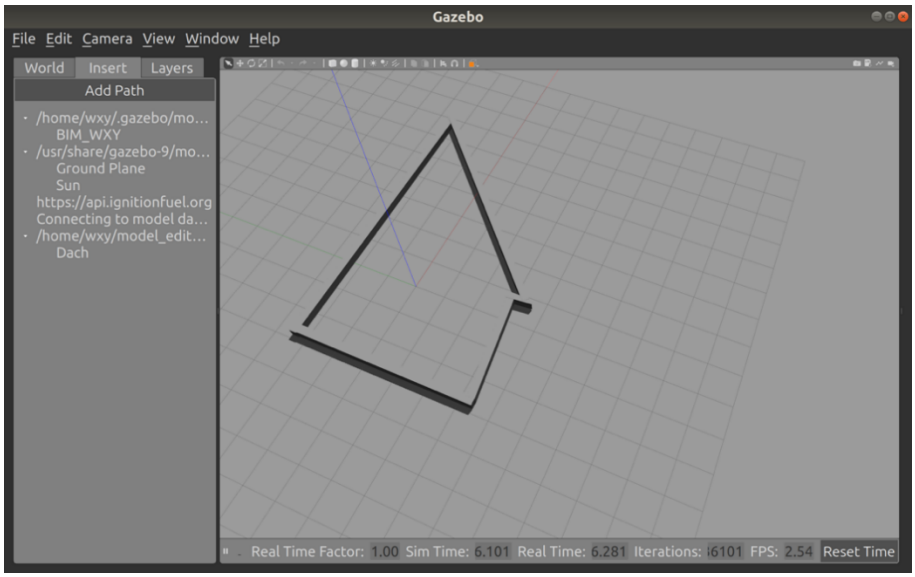

Abb. 4.7 Screenshot von Gazebo

Quelle: Eigene Modellierung

4.2.5. SLAM und Navigation

Wenn der Roboter während des Verlegens des Dachs autonom an vordefinierten Koordinaten und Reihenfolgen fahren will, muss er zunächst SLAM in einer Simulationsumgebung durchführen.

Mit ROS Launch wird die Simulationsumgebung in Gazebo aufgerufen, SLAM aktiviert und in Rviz angezeigt. Mit der Tastatur wird der Roboter so gesteuert, dass er langsam in der Simulationsumgebung fährt, damit das LIDAR alle festen Hindernisse in der Umgebung erfassen und eine Costmap erstellen kann. Die Koordinatensysteme von Costmap und Grundriss werden beim SLAM exakt übereinstimmend gehalten, so dass die Daten aus der CAD-Zeichnung ohne weitere Koordinatenumrechnung direkt für die spätere Zielpunktkonstruktion verwendet werden können. In der Costmap ist die Abweichung des TF-Koordinatensystem-Referenzpunkts des Roboters von der Ausgangsposition des TF-Koordinatensystems, wie in der Abbildung 4.8 gezeigt, der SLAM-Fehler, der im Idealfall 0 wäre. Der schwarze Teil ist die vom SLAM-Algorithmus identifizierte Wand des Dachs und wird in der Kostenkarte schwarz dargestellt, um eine 100%-ige Wahrscheinlichkeit für eine Kollision darzustellen. Fahren die Roboter mit der Tastatur so weit wie möglich über das Dach, bis die komplette Dachfläche in der Costmap grau dargestellt ist. Dann ist der SLAM-Prozess abgeschlossen und der

Befehl zum Speichern der Karte kann ausgeführt werden.

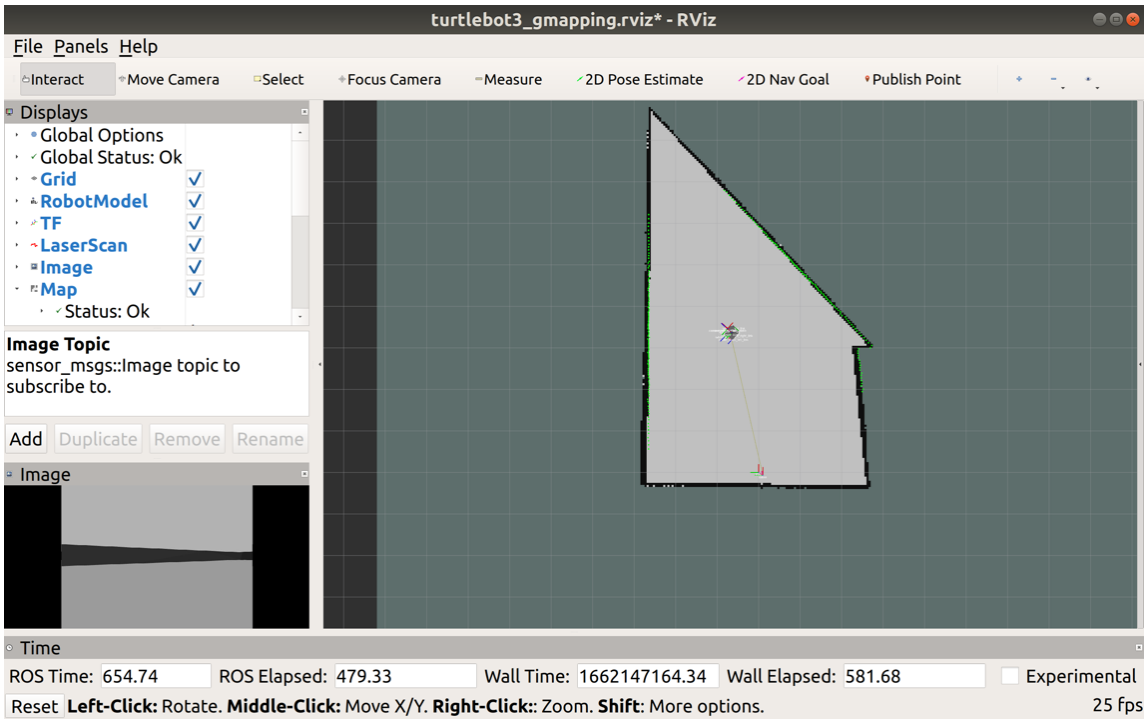

Abb. 4.8 SLAM auf Flachdach

Quelle: Screenshot von Rviz

Move_Base ist der Kernknoten von ROS Navigation, der sowohl globale als auch lokale Wege plant, indem er Map, amcl, LIDAR, Odometer und andere wichtige Knoten abonniert, und dann die Wege in Geschwindigkeitsinformationen für den Roboter übersetzt, und schließlich dem Roboter ermöglicht, zu navigieren.

In dieser Simulation wird der Zielpunkt über move_base/goal an den move_base-Knoten gesendet, indem ein Client erstellt wird, der den Roboter zu den gewünschten Positionen fahren lässt.

Da das Koordinatensystem von Move_Base über das Koordinatensystem Map festgelegt wird, ist der Ursprung der Koordinaten in der Map derselbe wie die Ausgangsposition des Roboters. Und weil das Flachdachmodell orthogonal zur ursprünglichen Position des Gazebo beim Entwickeln der Simulationsumgebung ist. Die Koordinaten der einzelnen Punkte und die Ausrichtung des Roboters können daher direkt mit Verlegeplan berechnet werden.

Damit wird das selbst geschriebene Programm ausgeführt, das den Server und den Client der Move_Base Action erstellt, und es wird eine Hauptschleife definiert, der Hauptzweck dieser Schleife besteht darin, eine goal-Variable zu initialisieren, sie in einen MoveBaseGoal-Typ umzuwandeln und den Frame der map zu verwenden, um

die frame_id der variablen goal zu definieren, und das goal die Positionskoordinaten und den Richtungswinkel des zuvor festgelegten Zielpunktes zu empfangen und schließlich die Zielpunktkoordinaten als MoveBaseGoal-Typ an die nächste move-Funktion zu senden. Zum Schluss wird der Zähler um 1 erhöht.

Danach werden die Koordinaten des in der Hauptschleife übergebenen Zielpunkts an den MoveBaseAction-Server gesendet, indem eine move-funktion definiert. Außerdem wird einen Timer von 3 Minuten gesetzt, so dass der Roboter, wenn er sich nach 3 Minuten immer noch nicht am angegebenen Zielpunkt erreicht, diesen verlässt und zum nächsten Zielpunkt weiterfährt. Bei erfolgreicher Erreichung des Ziels wird die Rückmeldung "Goal Success" ausgegeben.

Die Verlegung des Daches erfordert nämlich nicht nur einen Logistikroboter, der das Material und den Roboterarm zu den Zielpunkten transportiert, sondern auch einen Roboterarm oder ein anderes Gerät, das die Dämmplatten verlegen kann. Allerdings steht in dieser Arbeit nicht der Roboterarm im Fokus, dennoch wird am Ende der Funktion eine Wartezeit festgelegt. Sobald der Roboter den Zielpunkt erreicht hat, verbleibt er dort für 30 Sekunden, damit der "Roboterarm" die Verlegung der Dämmungen abschließen kann.

Das von rosgraph erzeugte strukturierte Baumdiagramm ist in Abbildung 4.09 dargestellt.

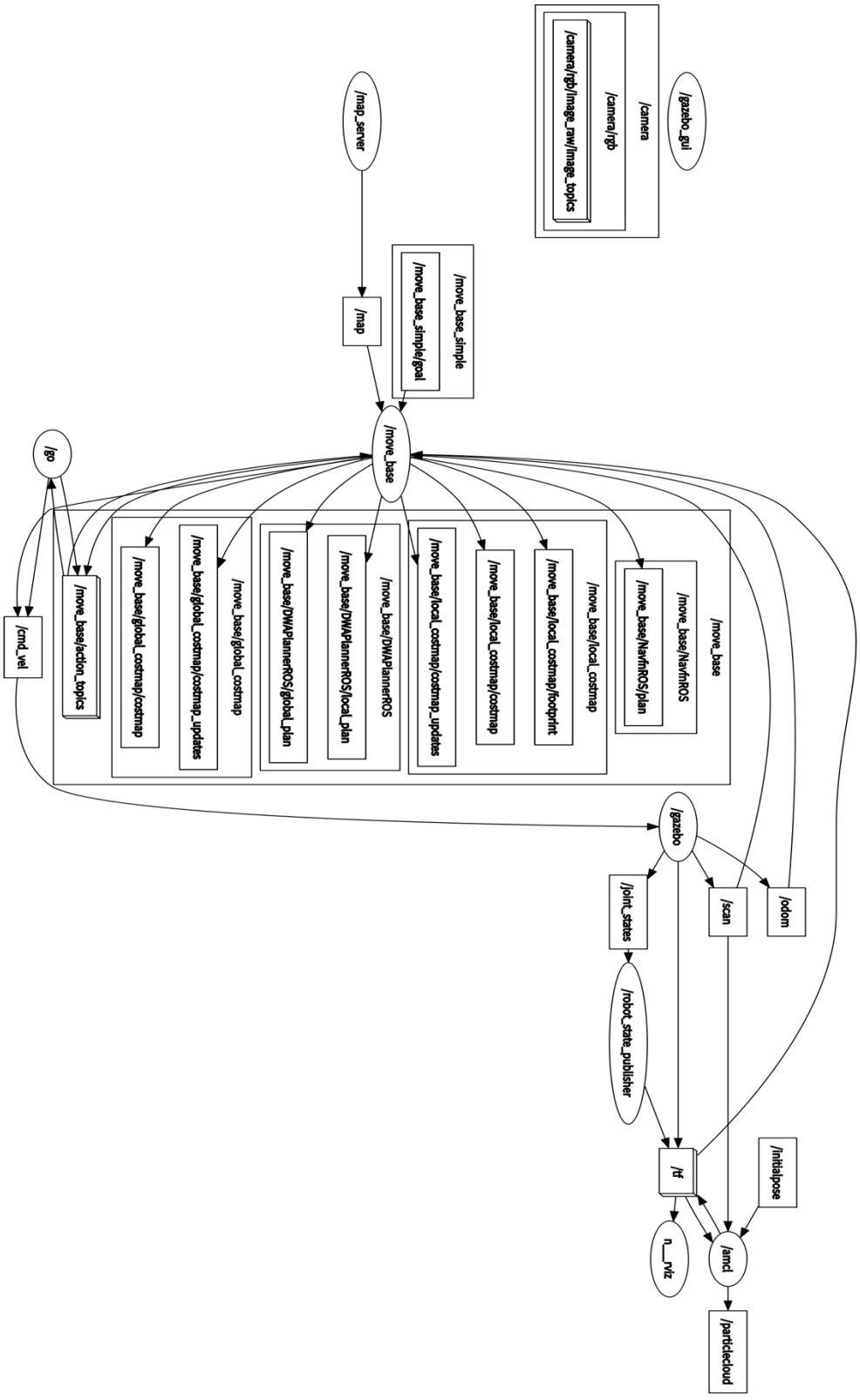

Abb. 4.9 Rosgraph
Quelle: Exportierung von ROS

4.2.6. Schwellenwerte für Canny-Operator

Die Untersuchungen wurden am 18. Juli 2022 in der Lagerhalle des Projektpartners Dachdeckermeister Claus Dittrich GmbH durchgeführt. Ziel der Experimente ist es, die Schwellenwerte im Canny-Operator für verschiedene Dämmplattenmaterialien zu bestimmen, so dass das Programm die Kanten der Dämmplatten besser erkennen kann.

Die Experimente werden wie folgt durchgeführt, wobei das auf dem Canny-Operator basierende Kantenerkennungsprogramm für verschiedene Dämmplattenmaterialien bei starkem Licht (1000 luminance), normalem Licht (600 luminance) bzw. schwachem Licht (250 luminance)zum Einsatz kommt. Durch die Einstellung des doppelten Schwellenwerts im Programm wird überprüft, ob dieser Schwellenwert dem Programm eine korrekte Bewertung des Zustands der Verlege ermöglicht und ob kein unerwünschtes Rauschen im Bild vorhanden ist. Für jedes Material wurden drei Versuche unter verschiedenen Beleuchtungsbedingungen durchgeführt, um den geeigneten Schwellenwert für jedes Material zu ermitteln, der dann in das Programm übernommen wurde. Die Ergebnisse der Experimente zeigen, dass jedes Material einen für sich geeigneten Schwellenbereich hat und dass der Schwellenbereich für jedes Material unterschiedlich ist. Die folgende Tabelle 4.1 zeigt die experimentellen Angaben. Die erste Stelle in der Tabelle steht für den untersten und die zweite Stelle für den obersten Schwellenwert.

Tab. 4.1 Daten des Experiments

Material	Stark Licht	Norma Licht	Schwach Licht
PU	1772/3773	1643/3085	1349/3436
EPS	947/3795	899/3672	1020/3780
Minerawolle	753/2747	753/2848	925/2776

Wie aus den Daten des Experiments hervorgeht, hat jedes der verschiedenen Materialien seinen eigenen Schwellenwert. Anhand der Schwellen kann der Operator die Kanten der Dämmplatten erkennen und feststellen, ob es dazwischen Fugen gibt. Auch die Lichtverhältnisse haben wenig Einfluss auf die Schwellenwerte, die bei gleichem Material und in jeder Umgebung in ungefähr dem gleichen Bereich liegen.

4.2.7. Dämmungen Identifizieren

Um die Messungen zu erleichtern, werden für die Messungen Schaumstoffblöcke aus demselben Material und in derselben Farbe wie das EPS verwendet. Bei der Längenmessung wird eine Längenreferenz im Bild platziert, aus der das Verhältnis von Pixeln zu Länge ermittelt wird. Die Koordinaten der Eckpunkte werden verwendet, um die Koordinaten des oberen und unteren Mittelpunkts des Rechtecks sowie die Länge des Rechtecks zwischen den beiden Seiten und dem Mittelpunkt zu berechnen.

Zunächst wird eine Identifikationsfunktion definiert, unter der dann eine Formelfunktion zur Bestimmung des Mittelpunkts zweier Punkte anhand ihrer Koordinaten definiert wird.

$$\begin{aligned}&\mathrm{Mittelpunkt}(A(x1,y1),\mathrm{B}(\mathrm{x2},\mathrm{y2})\\&=(\frac{x1+x2}{2},(\frac{(y1+y2}{2})\end{aligned} \tag{26}$$

Darüber hinaus müssen einige Parameter eingegeben werden, da es ein Referenzobjekt im Bild gibt, das zur Berechnung des Verhältnisses von Länge zu Pixel im Bild verwendet wird; daher müssen Länge und Breite des Referenzobjekts eingegeben werden.

Nach dem Einlesen des Bildes wird das Bild zunächst in eine Graustufenkarte umgewandelt, die dann mit einem Gauß-Filter von Bildrauschen befreit wird. Anschließend werden die Kanten in dem bearbeiteten Graustufenbild mit dem Canny-Operator identifiziert. Dann muss das identifizierte Graustufenbild erneut erodiert und erweitert werden, damit die identifizierten Kanten zu einer vollständigen Kontur geschlossen werden. Schließlich wird die FindContours-Funktion verwendet, um die Kontur der identifizierten Objekte zu bestimmen.

Danach wird mit der Schleifenfunktion jeder Kontur im Bild durchsucht, wobei zunächst das Bildrauschen entfernt wird, wenn der Fläche des Konturbereichs zu klein ist.

Anschließend werden die vier Scheitelpunkte des Konturs gesucht und die Scheitelpunkte sowie die gewünschten Mittelpunkte im Bild markiert. Aus den Koordinaten wird die Anzahl der Pixel der beiden Seiten des Rechtecks und zwischen den beiden Mittelpunkten berechnet. Die tatsächliche Länge wird dann auf der

Grundlage des Verhältnisses zwischen den zuvor berechneten Pixeln und der Länge im Bild berechnet. Am Ende wird der durchschnittliche Wert der Längen der drei Seiten im Bild angezeigt.

In Abbildung 4.10 werden die Abmessungen der Referenz entsprechend den zuvor eingegebenen Daten im Bild angezeigt.

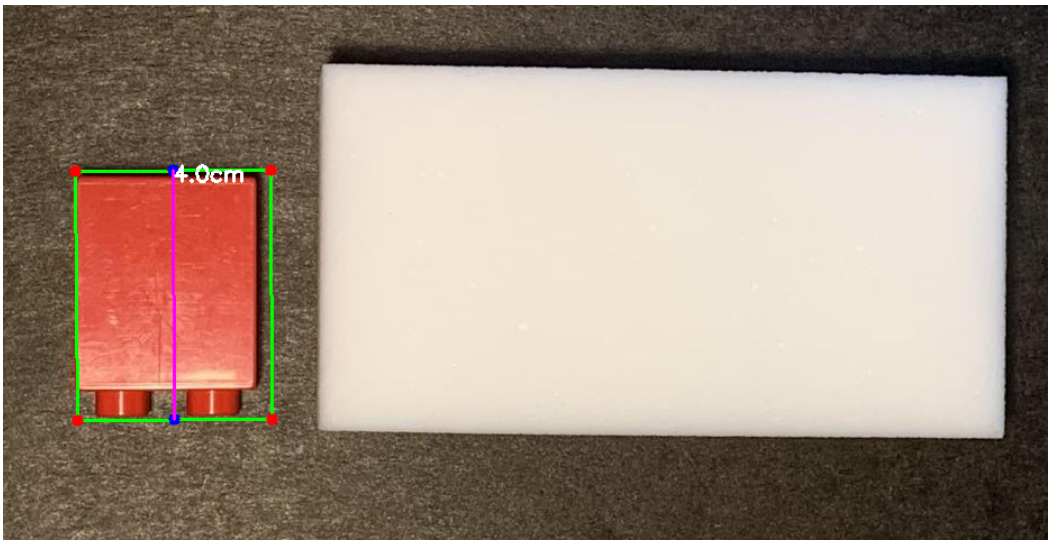

Abb. 4.10 Identifizierung von Referenz
Quelle: ScreenShot

4.2.8. Fugen Identifizieren

Bei der Verlegung von Dachdämmplatten sollten die einzelnen Platten fugenlos und eng aneinandergelegt werden. Diese Regel kann somit bei der Erkennung der Dämmungen verwendet werden, indem der Canny-Operator zur Kantenerkennung des Bildes eingesetzt wird. Wenn in der Mitte der beiden Dämmplatten keine Kanten (Fugen) erkannt werden, sind die Dämmplatten nachweislich in der richtigen Verlegeposition.

Die verschiedenen Dachdämmplatten haben unterschiedliche Materialien, Reflexionen und Texturen. Daher ist es notwendig, die in früheren Tests gemessenen Schwellenwerte für jedes Material als Parameter für die Fugenprüfung zu verwenden.

4.2.9. Simulationsergebnis

Die Simulationsergebnisse werden im Folgenden in Bezug auf den Arbeitsablauf eines Roboters auf einem Flachdach dargestellt.

Aufgrund der geringen Größe des reduzierten Flachdachs und der Tatsache, dass die Koordinaten des in Gazebo geladenen Roboters der Koordinatenursprung sind, fallen der Ursprung des Odom- und des Map-Koordinatensystems des Roboters zu Beginn zusammen. Auch weil es sich um eine Simulationsumgebung handelt, ist alles idealisiert. Bei dieser Flachdachsimulation muss AMCL nicht vor dem Start durchgeführt werden.

Sobald der Wagen den ersten Punkt erreicht hat, wird über Move_Base eine Nachricht „Goal Succeeded“ zurückgeschickt, um mitzuteilen, dass der Zielpunkt erfolgreich erreicht wurde und die Messung begonnen hat (Siehe Abbildung 4.11).

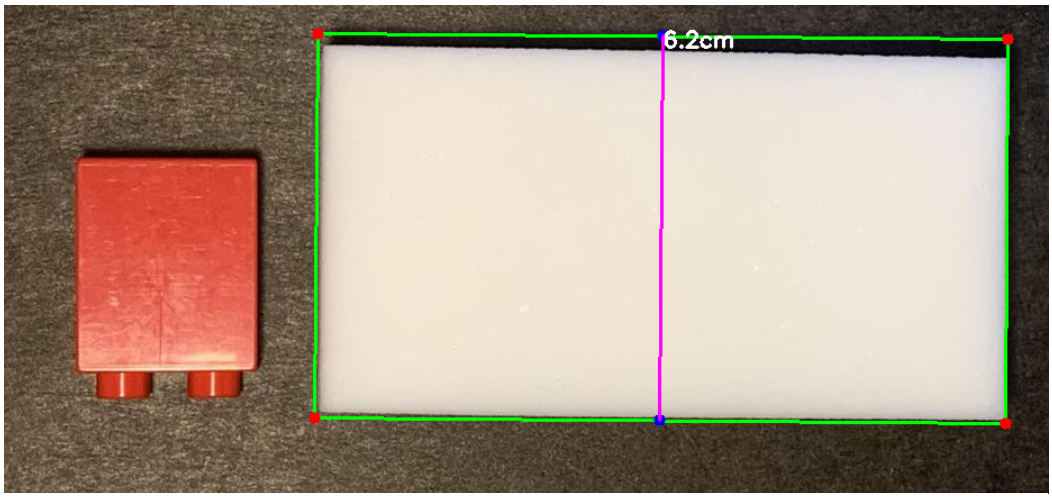

Abb. 4.11 Identifizierung von Dämmplatten
Quelle: ScreenShot

Wenn die Messung abgeschlossen ist, fährt der Roboter zum festgelegten zweiten Zielpunkt (Siehe Abbildung 4.12). Der globale Weg zum zweiten Zielpunkt wird durch den globalen Wegplaner geplant, die Umgebung wird durch LiDAR gescannt und eine neue Strecke wird durch den lokalen Wegplaner geplant, sofern ein Hindernis erkannt wird. Der AMCL-Algorithmus verhindert, dass das Odom-Koordinatensystem des Roboters abdriftet, während der Roboter fährt.

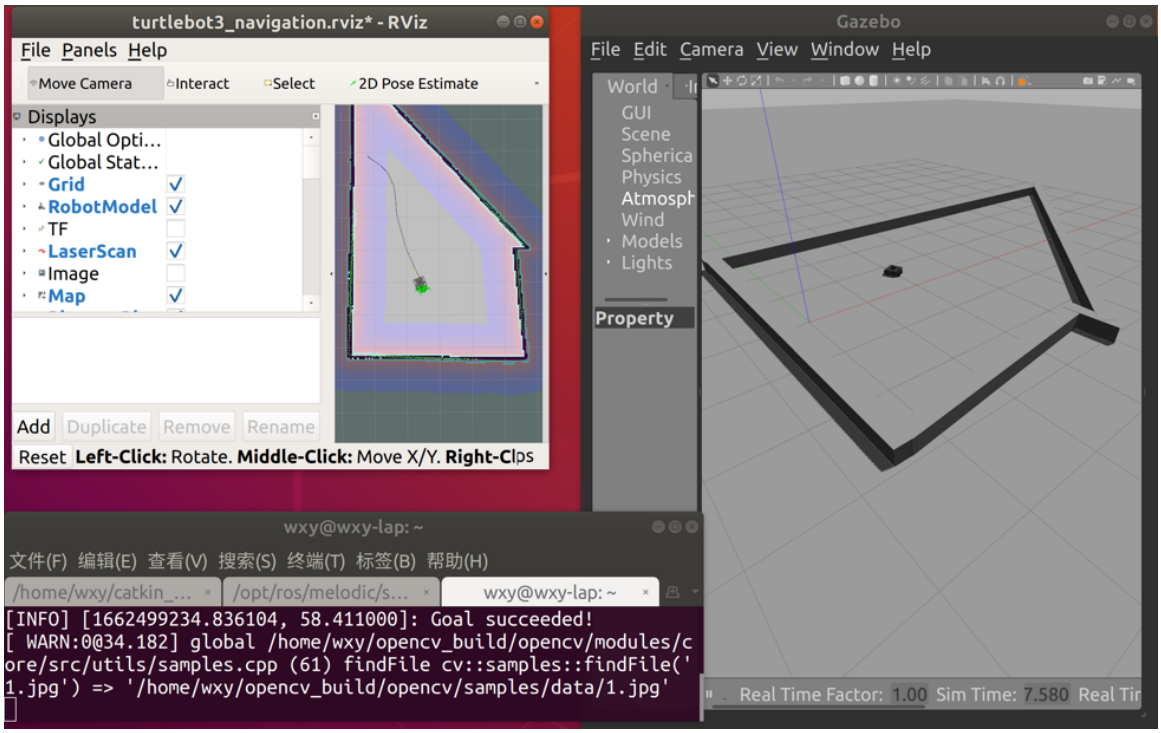

Abb. 4.12 Fahren zur erste Verlegungsposition
Quelle: Screenshot

Bei Erreichen des zweiten Zielpunktes wird das Bild ausgelesen und die Fuge identifiziert. Wird eine Fuge festgestellt, ist die Dämmplatteninstallation fehlerhaft, wird keine Fuge festgestellt, ist die Dämmplatte korrekt installiert. Alle anderen Schritte sind ein Zyklus der Schritte 2 und 3, so dass es nicht nötig ist, sich allzu ausführlich damit zu befassen.

Deshalb sind die Ergebnisse der Identifizierung erfolgreicher und nicht erfolgreicher Dämmplatten durch den Canny-Operator einmal in Abbildung 4.13 und Abbildung 4.14 dargestellt.

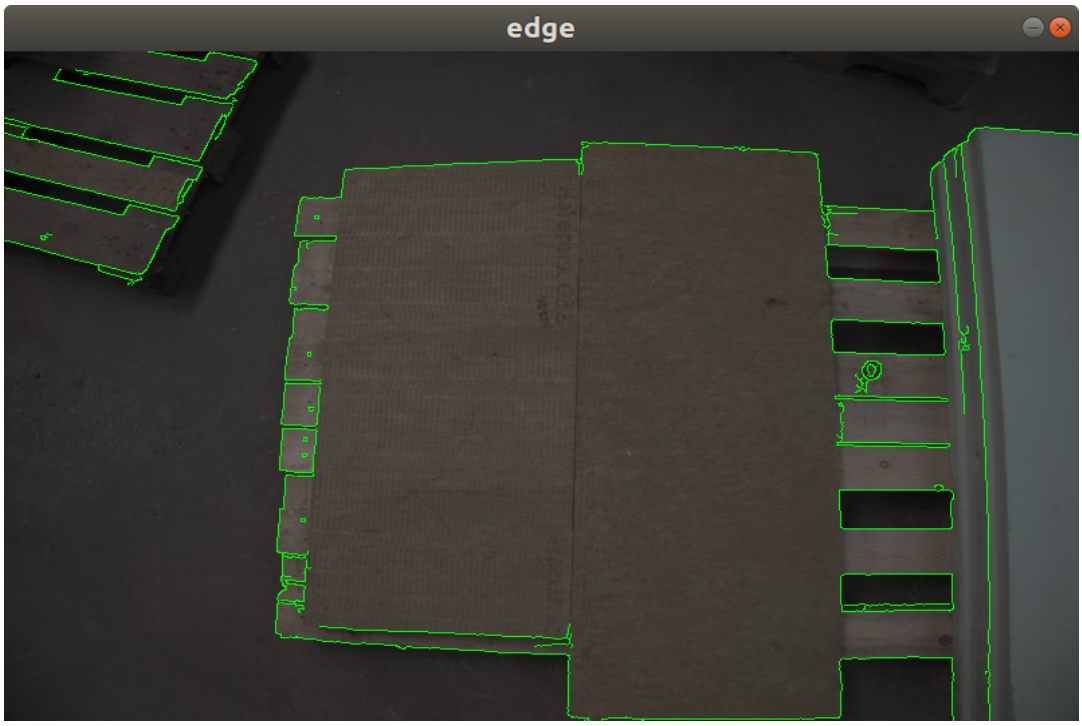

Abb. 4.13 Prüfung der Fugen ohne Baufehler

Quelle: Lagerhalle bei Dachdeckermeister Claus Dittrich GmbH

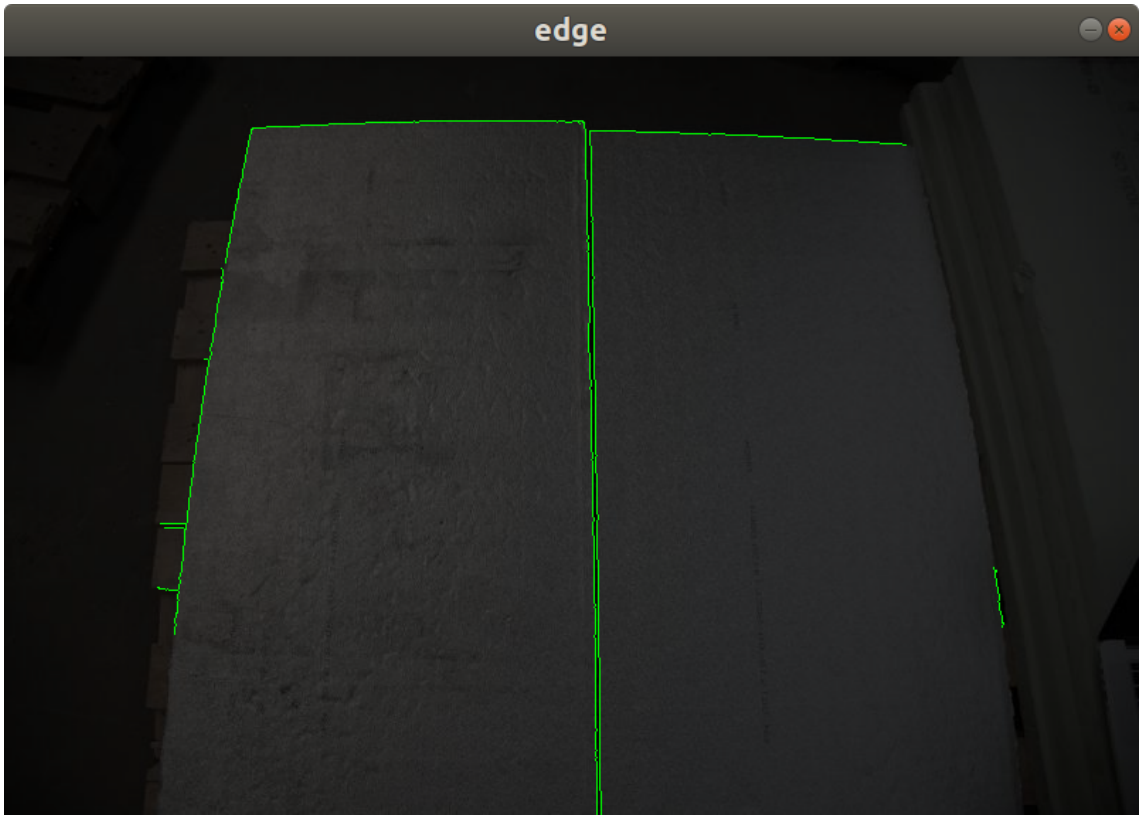

Abb. 4.14Prüfung der Fugen mit Baufehler

Quelle: Lagerhalle bei Dachdeckermeister Claus Dittrich GmbH

4.3. Kooperation der Simulation und Realität

In diesem Abschnitt wird untersucht, ob es möglich ist, die Simulationsumgebung mit Hilfe von SLAM mit der realen Situation zu kombinieren. Der Hauptansatz bestand darin, mit Hilfe des Gmapping-SLAM-Algorithmus in der Simulationsumgebung eine Karte des Instituts zu erstellen. Der Roboter wurde dann in der realen Welt eingesetzt und das Navigations-Paket wurde direkt anhand der in der Simulationsumgebung erstellten Karte gestartet. In Tests wurde überprüft, ob der Roboter die AMCL-Lokalisierung, die Erkennung fester und temporärer Hindernisse und die entsprechende Wegplanung innerhalb des Instituts durchführen kann.

4.3.1. Entwicklung der Simulationsumgebung

In dieser Arbeit kann die aus Revit exportierte Collada-Datei noch einmal mit Blender angepasst werden. Beispielsweise sind die in Abbildung 4.15 mit rot dargestellten Trennwänden im BIM-Modell nicht vorhanden, werden aber in der Simulation ebenfalls als feste Hindernisse verwendet, im Gegensatz dazu werden die mit blau dargestelltem Tisch und Stühle als temporäre Hindernisse auf der Baustelle betrachtet und nicht in die Simulationsumgebung aufgenommen.

Abb. 4.15 Institut für Bauinformatik

Quelle: Eigene Aufnahme

In Abbildung 4.15 wurde das zuvor exportierte Modell des Instituts angepasst, indem z. B. alle Türen entfernt und feste Hindernisse hinzugefügt wurden. Das angepasste Modell kann dann zur späteren Verwendung wieder als Collada-Datei exportiert werden. Das angepasste 3D-Modell wurde dann auf die gleiche Weise exportiert, um mit Gazebo eine Simulationsumgebung für den angepassten Roboter zu schaffen (Siehe Abbildung 4.17).

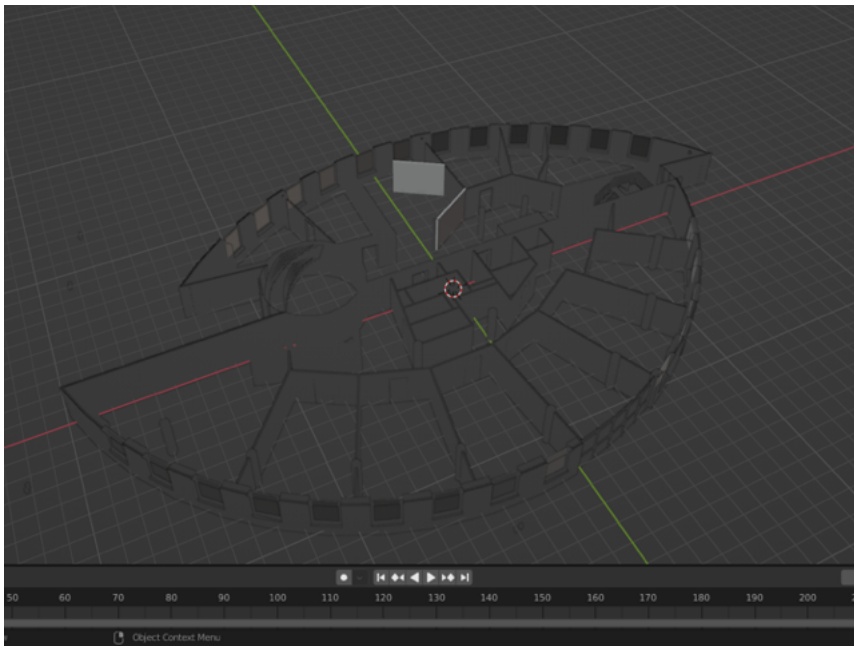

Abb. 4.16 Screenshot von Blender
Quelle: Eigene Modellierung

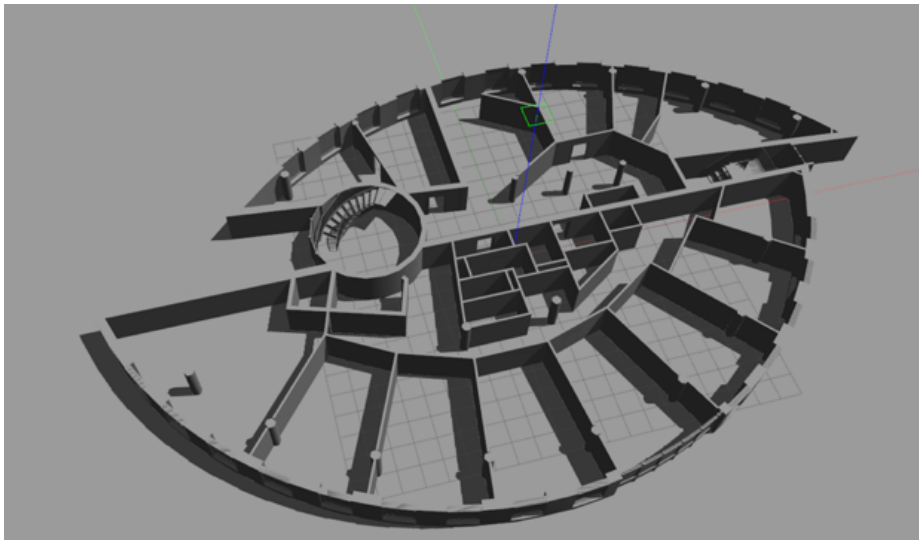

Abb. 4.17 Screenshot von Gazebo
Quelle: Eigene Mdellierung

4.3.2. SLAM in der Simulationsumgebung

Mit Launchdatei von ROS wurden Gazebo mit der geladenen Roboter- und Simulationsumgebung und SLAM mit dem geladenen Gmapping-Algorithmus zusammen mit der Rviz-Software gestartet, wie in Abbildung 4.14 und Abbildung 4.18 dargestellt.

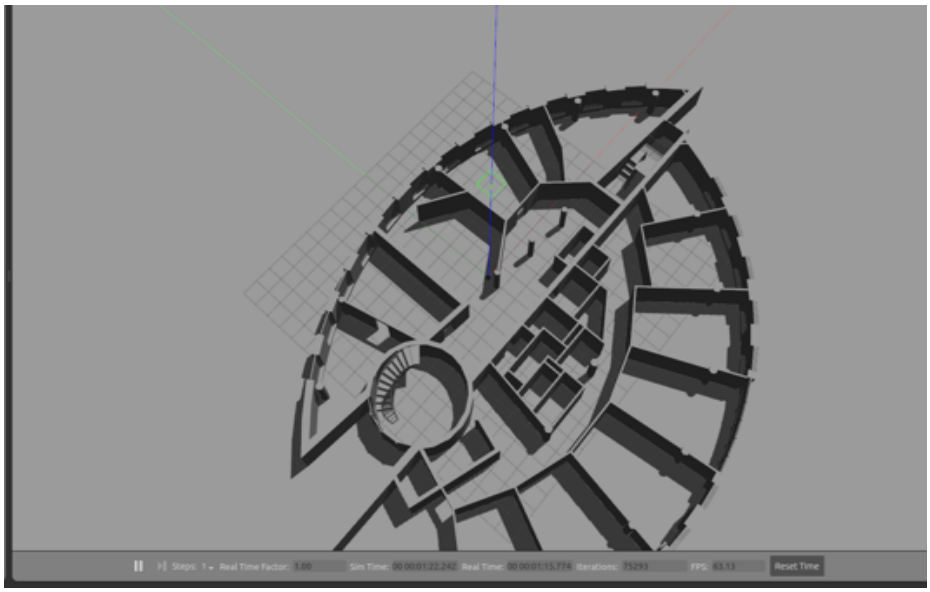

Abb. 4.18 Turtlebot3 in Gazebo
Quelle: Eigene Modellierung

Das SLAM kann durch Aktivierung des Softwarepakets für automatisches Fahren des Roboters durchgeführt werden, mit dem der Roboter in der Simulationsumgebung so lange vorwärtsfährt, bis er auf ein Hindernis trifft, dann in einem zufälligen Winkel abbiegt und weiterhin in einer geraden Strecke fährt. Dieser Vorgang wird wiederholt, bis die SLAM-Erstellung abgeschlossen ist (Robotis 2022b). Der Vorteil dieses Ansatzes besteht darin, dass andere Aufgaben erledigt werden können, während der Roboter SLAM durchführt, und dass nach im Hintergrund Abschluss des SLAM der Roboter die Karte speichern kann. Der Nachteil ist, dass bei komplexeren Simulationsumgebungen, wie z. B. im Institut für Bauinformatik, die Erstellung der Karte durch den Roboter möglicherweise sehr viel Zeit erfordert, was wiederum zu Ineffizienzen führt. Alternativ kann SLAM auch durch das Bedienen der Tastatur zur Steuerung des Roboters in der Simulationsumgebung und das anschließende Speichern der Karte durchgeführt werden (Robotics 2022b). Die

folgende Abbildung 4.19 zeigt auch den SLAM-Prozess.

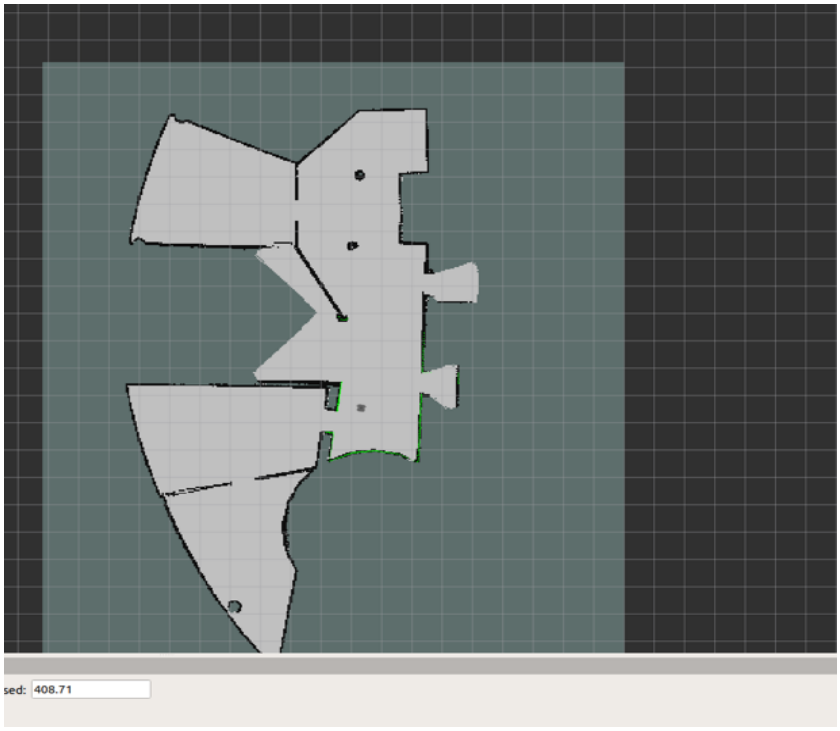

Abb. 4.19 SLAM Prozess
Quelle: Eigene Modellierung

4.3.3. Navigation des Roboters in reale Umgebung

In Abbildung 4.20 dargestellt, dass der Roboter zunächst grob an dem Nullpunkt platziert wird, an dem der Roboter während des vorherigen SLAMs geladen wurde. Mit AMCL muss die Platzierung des Roboters nicht besonders genau sein. AMCL überträgt die Daten des Laserscans an den Odometer des Roboters, und durch den Filter ist es möglich, die Position des Roboters relativ zur Karte zu verfolgen, so dass der Roboter seine Position in der Karte in Echtzeit aktualisieren und anpassen kann.

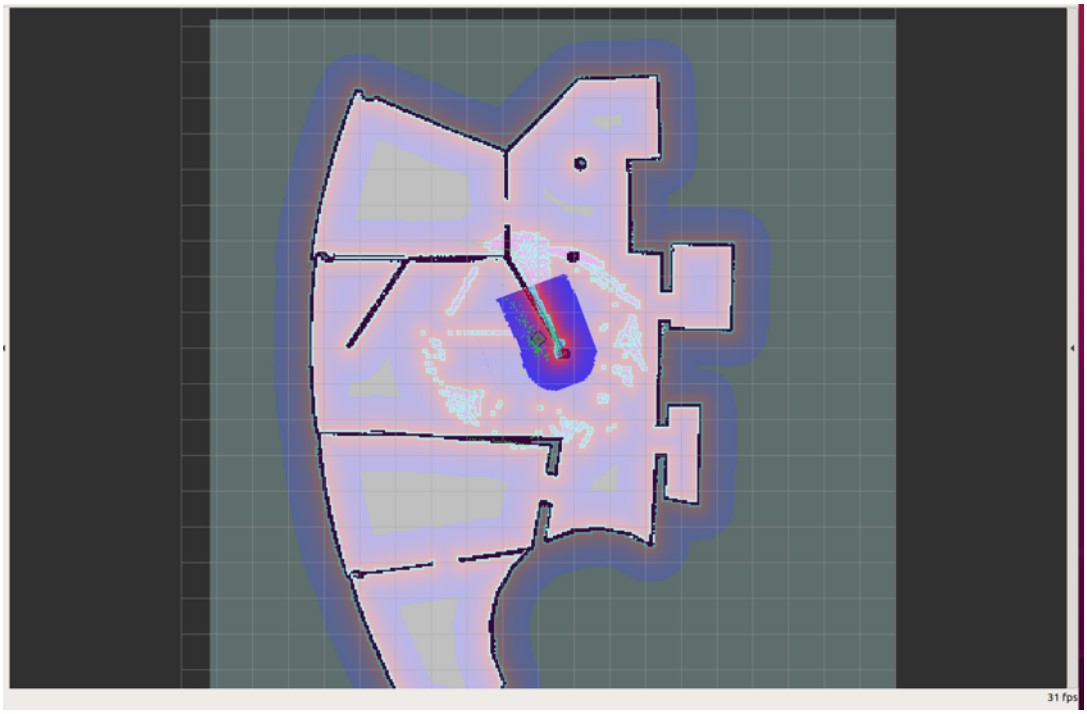

Abb. 4.20 AMCL
Quelle: ScreenShot

Der dunkelblaue Teil von Abbildung 4.21 ist die Local-Costmap des Roboters, und der hellblaue Teil enthält die Global-Costmap. Wenn keine globale Kostenkarte vorhanden war, würde der Roboter zunächst eine gerade Linie zu diesem Zielpunkt fahren, solange kein Hindernis erkannt wurde, und dann den Weg ändern, wenn ein Hindernis erkannt wurde. Anhand der vorherigen Gittergraukarte von SLAM und der Global-Costmap, die bei der Navigation verwendet wird, wird der Roboter unabhängig von seiner Position auf ein festes Hindernis reagieren.

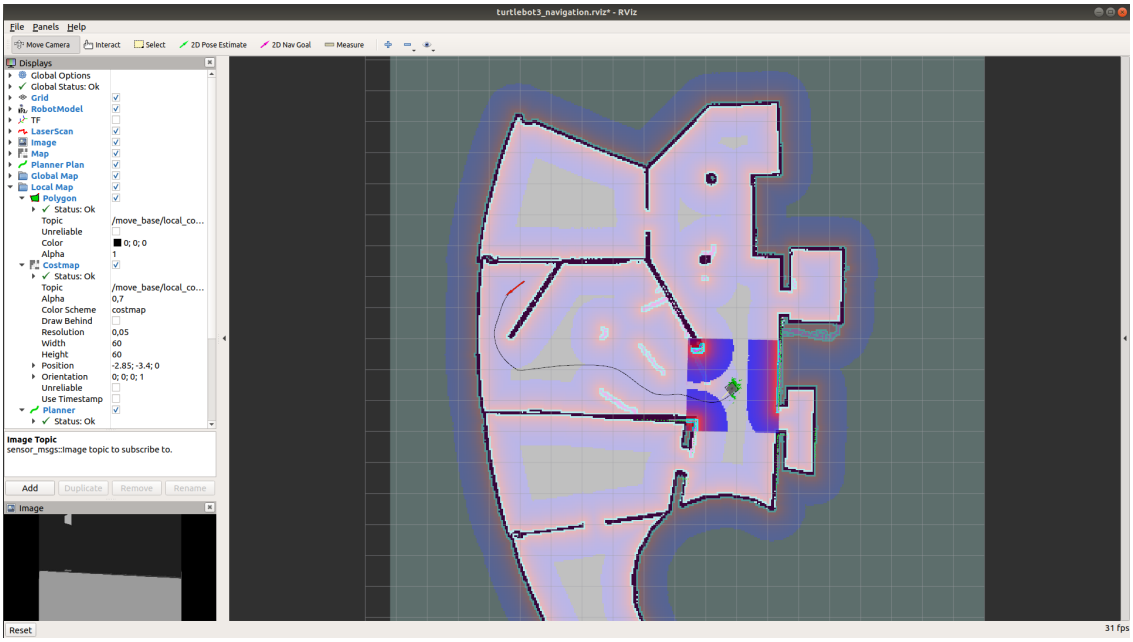

Abb. 4.21 Navigation mit Global-Costmap

Quelle: ScreenShot

In der Local-Costmap des Roboters in Abbildung 4.22 wird ein temporäres Hindernis erkannt, das sich nicht in der Simulationsumgebung befindet, und der Roboter plant seine Route auf der Grundlage der Position des Hindernisses.

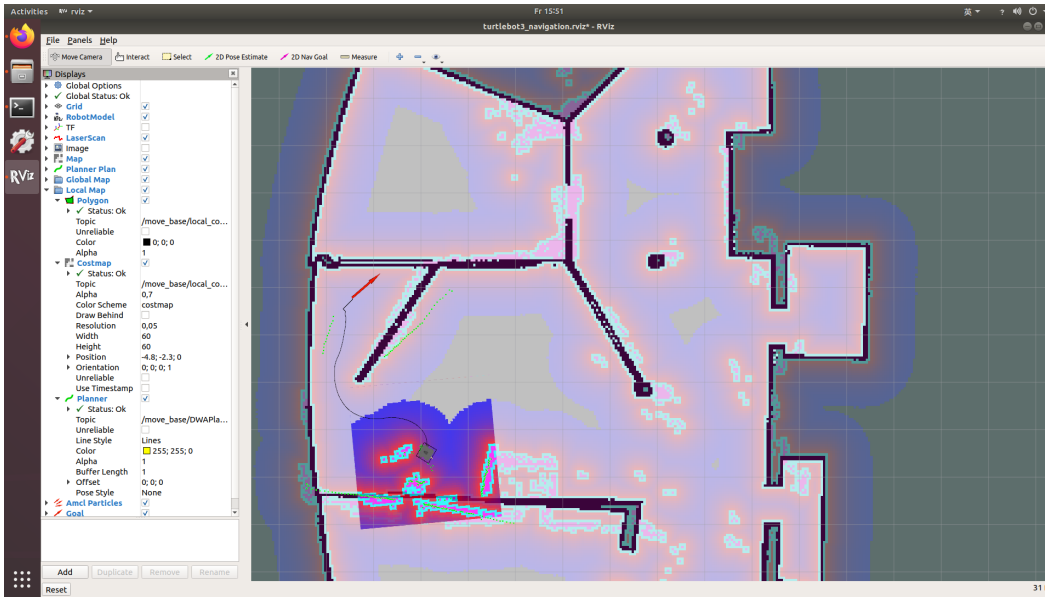

Abb. 4.22 Navigation mit Local-Costmap

Quelle: ScreenShot

4.4. Beurteilung der Simulationen

Die Ergebnisse der beiden Simulationen entsprechen perfekt den angestrebten Zielen. In der ersten Simulation wird der Roboter durch autonomes Fahren nacheinander zum geplanten Zielpunkt gefahren. Da es in der ersten Simulation keine temporären Hindernisse gibt, kann der Roboter das Ziel mit der globalen Costmap und der globalen Wegplanung relativ reibungslos erreichen. Nach Erreichen des Ziels gibt ROS-Action eine Rückmeldung, dass der Roboter das Ziel erfolgreich erreicht hat. Gleichzeitig ist der Roboter in der Lage, die Dämmplatten entsprechend den im Programm vorgesehenen Zählern zu identifizieren oder zu überprüfen. In beiden Fällen entsprechen die Simulationsergebnisse weitgehend den Erwartungen: In der Identifizierungsphase liegt der Fehler bei der Messung der Länge innerhalb von 5 %. Auch die Überprüfung, ob die Dämmplatten erfolgreich verlegt wurden, entspricht den gestellten Anforderungen.

In der zweiten Simulation erfasste und tastete der Roboter die Karte der Simulationsumgebung sehr effizient ab. Würde der Roboter SLAM direkt in der realen Umgebung statt auf diese Weise durchführen, würde es viel länger dauern. Außerdem würde die Karte nach SLAM aufgrund von temporären Hindernissen zu viele nutzlose oder falsche Informationen enthalten. Auch bei der Verwendung von Karten zur Navigation in realistischen Umgebungen zeigt der Roboter zufriedenstellende Leistungen. Es kann nicht nur eine generelle Fahrroute auf der Grundlage globaler Costmap und globaler Wegplanung planen, sondern auch eine automatische temporärer Hindernisvermeidung auf der Grundlage, von LiDAR erfasster Hindernisse mit Hilfe lokaler Costmap und lokaler Wegplanung daüber durchführen.

Zusammenfassend lässt sich sagen, dass die Simulationen die Pläne und Erwartungen dieser Arbeit erfüllen. Die Verwendung des Roboters und der Methoden kann weiterhin in praktischen Projekten eingesetzt werden. Im nächsten Kapitel werden die Ergebnisse der beiden Simulationen verknüpft und es werden Beispiele implementiert

5 Implementierung

Aus der Zusammenfassung der Ergebnisse der ersten und zweiten Simulation wissen wir, dass der Roboter eine SLAM-Karte in der auf dem BIM-Modell basierenden Simulationsumgebung erstellen und die daraus erstellte Karte für die Navigation und das Ausweichen vor temporären Hindernissen in der realen Umgebung verwenden kann. Wir wissen auch, dass die Koordinaten des Zielpunkts über den Move_Base-Knoten an den Roboter gesendet werden können, und der Roboter wird mithilfe des globalen Bahnplaners einen globalen Weg planen, der an dieser Koordinate im Koordinatensystem Map endet.

Durch die Kombination der ersten und zweiten Simulation wird auf diese Weise ein komplettes Beispiel in einer realen Umgebung umgesetzt. Der Roboter am Institut BauInformatik soll den Weg und die Reihenfolge der Fahrt entsprechend den vereinfachten Zielpunkten des Verlegeplans verfolgen und die in der zweiten Simulation erstellte Karte zur Navigation nutzen. Außerdem können die Bilder von Roboter gelesen werden, um Dämmplatten mit Hilfe von OpenCV zu vermessen oder zu überprüfen, ob sie korrekt verlegt sind.

5.1. Rahmenbedingungen

Für die Implementierung dieses Beispiels sollten die folgenden Bedingungen erfüllt sein, um sicherzustellen, dass das Beispiel vollständig umgesetzt werden kann

- Der Arbeitsablauf des Roboters ist mit dem des Roboters in der ersten, nämlich der Flachdachsimulation gleich.
- Da die Simulationsumgebung des Instituts BauInformatik bereits in der zweiten Simulation fertiggestellt wurde (Siehe Abschnitt 4.3.1.), ist es nicht notwendig, die Simulationsumgebung für diese Beispielimplementierung neu zu entwickeln.
- In gleicher Weise wurde in der zweiten Simulation ein SLAM-Prozess auf der Simulationsumgebung des Instituts BauInformatik durchgeführt und bereits eine Gitterkarte erstellt (Siehe Abschnitt 4.3.2.). In dieser Beispielimplementierung ist es nicht mehr notwendig, SLAM in der Simulationsumgebung durchzuführen, und die Karte kann direkt zur Navigation genutzt werden.

- In der Beispielimplementierung wird der in der ersten Simulation erarbeitete Verlegeplan vereinfacht, indem er direkt befolgt und sieben Zielpunkte definiert, die der Roboter anfahren soll. Einer davon ist der Ursprungspunkt N, an dem das Material gelagert wird. Die anderen sechs Punkte sind A1, B1, B2, C1, C2 und C3, wobei die verschiedenen Buchstaben für die verschiedenen Kategorien von Dämmplatten stehen. Auf dem Boden des BauInformatik-Instituts werden Etiketten mit Informationen zu den realen Koordinaten angebracht, um zu überprüfen, ob der Roboter den Zielpunkt tatsächlich erreicht hat.
- Aufgrund der Beschädigung der 3D-Kamera werden zuvor aufgenommene Fotos vom Computer direkt eingelesen und die Funktion mit OpenCV zu vervollständigen.
- Der in diesem Kapitel für den Roboter verwendete Code ist fast identisch mit dem der ersten Simulation, außer der Anzahl und den Koordinaten der Zielpunkte und der Anzahl der entsprechenden Punkte, die durch Computer Vision zu erfassen sind, zu unterscheiden. Daher wird der Code für die vollständige Beispielimplementierung in den Anhang aufgenommen, und der Code für die erste Simulation wird teilweise in Anhang II aufgeführt.

5.2. Zielpunktkoordinaten

Der Roboter wird so eingestellt, dass er in der in Abbildung 4.4 gezeigten Reihenfolge fährt, mit insgesamt sieben Punkten, wobei N die Ursprungskoordinate **(0, 0, 0)** ist und die Richtung von Punkt N als **3*Pi/2** auf der Grundlage der Anfangsrichtung des Roboters berechnet wird. Die übrigen sechs Punkte befanden sich zuvor in einem Abstand von 1 m auf der x- und y-Achse, und die Koordinaten des Punktes A1 wurden auf **(-0,30, 3,50, 0)** in Richtung **Pi/2** gesetzt, so dass die Koordinaten der Punkte B1-C3 wie folgt lauteten: (**-0.30, 2.50, 0**), **(0.70, 2.50, 0)**, **(-0.30, 1.50, 0)**, **(0.70, 1.50, 0)**, **(1.70, 1.50, 0)**. Alle fünf Punkte in der Richtung sind Pi/2, außer C3, der 0 ist.

Abb. 5.1 Zielpunkten der Roboter
Quelle: Eigene Aufnahme in Institut für BauInformatik

5.3. Durchführung

Um die Wichtigkeit des AMCL zwischen dem BIM und dem Roboter zu bestätigen, wurde der Roboter später weiter von der korrekten Ursprungsposition entfernt platziert, um den Roboter zu starten, so dass die Koordinatensysteme von Odom und Map nicht übereinstimmten. Wie in Artikel 3.4.3. erwähnt, ist es die Stabilität der einzelnen Koordinatensysteme, die es dem Roboter ermöglicht, in der BIM besser zu funktionieren. Das AMCL gewährleistet die Stabilität zwischen den Koordinatensystemen.

Zunächst werden der Roboter und der Computer im selben WLAN über SSH miteinander verbunden. Auf der Computerseite wird Roscore als Master gestartet und der Computer versorgt Turtlebot3 über sein eigenes, lokal installiertes ROS-Paket mit Rechenleistung. Turtlebot3 funktioniert nur als Publisher seiner eigenen Sensoren und als Subscriber von Bewegungsbefehlen.

5.3.1. AMCL

In Abbildung 5.2 ist zu sehen, dass die schwarzen Teile die Stellen in der Karte darstellen, an denen der Roboter nicht fahren kann, d.h. feste Hindernisse. Die AMCL-Partikelpunktwolke ist sehr spärlich um den Roboter herum verteilt, und die Unschärfe der Punktwolke repräsentiert die Ungenauigkeit der Position des Roboters. Zu diesem Zeitpunkt ist die Funktion noch nicht konvergiert. Das Bild zeigt eine grüne Linie mit einem Kreisbogen oben rechts vom Roboter, die das vom LIDAR des Roboters erkannte Hindernis darstellt, nämlich die Wand mit dem Pfeiler rechts vom Roboter. Das Koordinatensystem von Odom und Map stimmt nicht überein, da sich der Roboter nicht in der richtigen Ausgangsposition befindet.

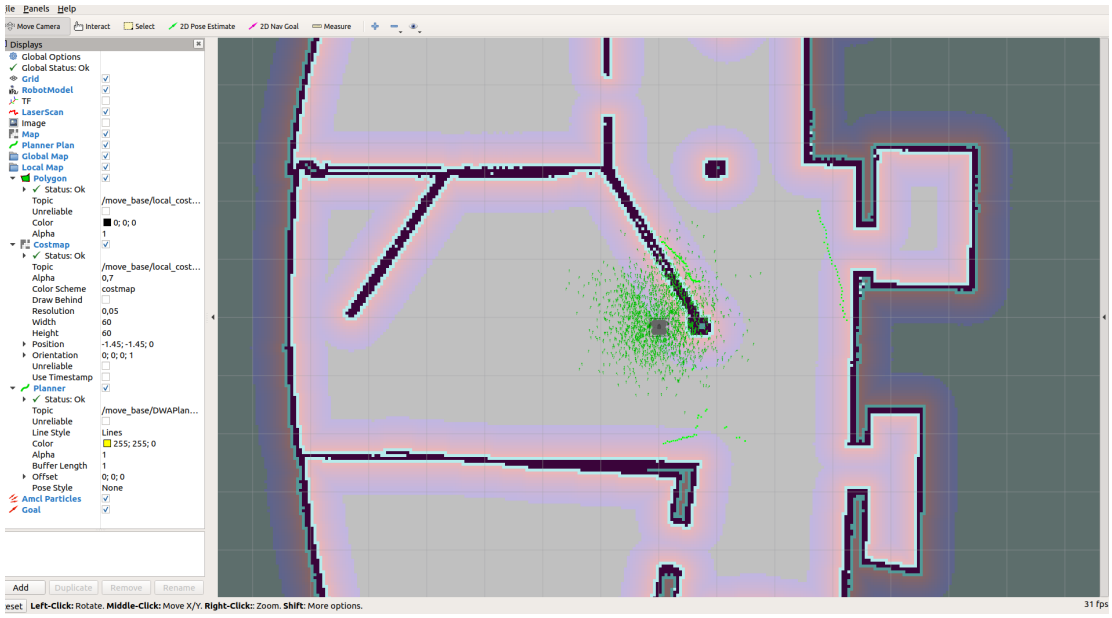

Abb. 5.2 AMCL

Quelle: ScreenShot

Der Knoten turtlebot3_teleop_key wird ausgeführt und der Roboter wird vorübergehend über die Tastatur gesteuert. Nach längerer Fahrt bzw. nachdem das LIDAR mehr Informationen über die Umgebung erfasst hat, beginnt die Funktion zu konvergieren, und die AMCL-Punktwolke wird immer dichter, bis die Punktwolke schließlich den gesamten Roboter erfasst hat. Zu diesem Zeitpunkt wurde festgestellt, dass die vom LIDAR erfassten Hindernisse in der Umgebung mit den Hindernissen auf der Karte übereinstimmten wie in der Abbildung 5.3. Dies bedeutet, dass der Ursprung des Koordinatensystems von Odom und Map, die

bereits fehlerhaft waren, nun durch die AMCL wieder identisch sind. Ohne den AMCL-Algorithmus wäre die Navigationsfunktion des Roboters im Falle der oben genannten Situation oder der in den Abschnitten 3.4.3 und 3.5.1 erwähnten Drift des Koordinatensystems von Odom völlig nutzlos. Der Roboter kann weder zu dem in Rviz markierten Zielpunkt noch zu den per ROS-Aktion an Move_Base übergebenen Koordinaten fahren.

Abb. 5.3 Erfolgreich AMCL
Quelle: ScreenShot

5.3.2. Fahren zum N Punkte und Dicke Abmessung

Nachdem der Roboter von der AMCL positioniert wurde, beginnt der Kernteil der Implementierung dieses Beispiels, indem der Code geladen wird, der für die Durchführung des Projekts geschrieben wurde. Da es im Bereich der visuellen Messung durch OpenCV ein Referenzobjekt gibt, müssen dessen Abmessungen mit dem Befehl zum Öffnen des Programms eingegeben werden. Daher wird beim Programmstart der Parameter - -laenge 4 nach dem Programm hinzugefügt, der die Länge des Referenzobjekts von 4 cm darstellt.

Nun folgt der Roboter dem festgelegten Arbeitsablauf zu Punkt N. Dazu nutzt er die globale Wegplanung, erfasst Hindernisse auf dem Weg über LIDAR und verwendet einen lokalen Wegplaner, um temporäre Hindernisse zu umgehen, falls entsprechende Hindernisse erkannt werden (Siehe Abbildung 5.4).

Abb. 5.4 Roboter fährt zum N Punkte

Quelle: Eigene Aufnahme in Institut für BauInformatik

Der Roboter erreicht Punkt N und gibt über ROS Action die Rückmeldung, dass er das angegebene Ziel erfolgreich erreicht hat. Unmittelbar darauf wird das Bild zur Längenmessung gelesen. Wie in Abbildung 5.4 zu sehen ist, wurde die Länge mit 12 cm gemessen, was genau dem tatsächlichen Wert entspricht.

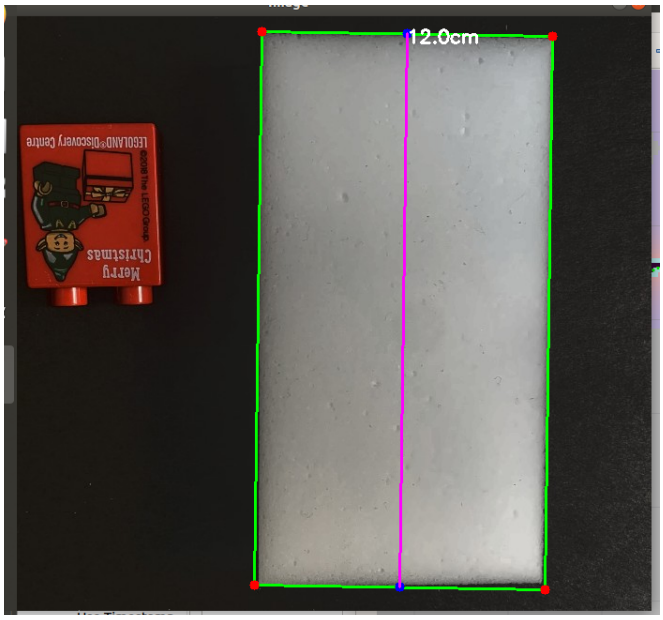

Abb. 5.4 Längeabmessung

Quelle: ScreenShot

5.3.3. Fahren zum A1 Punkte und Fugen Identifizierung

Nach der Bestätigung, dass die Länge korrekt ist, verlässt man das Messfenster und fährt sofort zum nächsten Zielpunkt A1, wie in Abbildung 5.5 zu sehen ist, der dem auf dem Bildschirm in Rviz angezeigten Weg folgt.

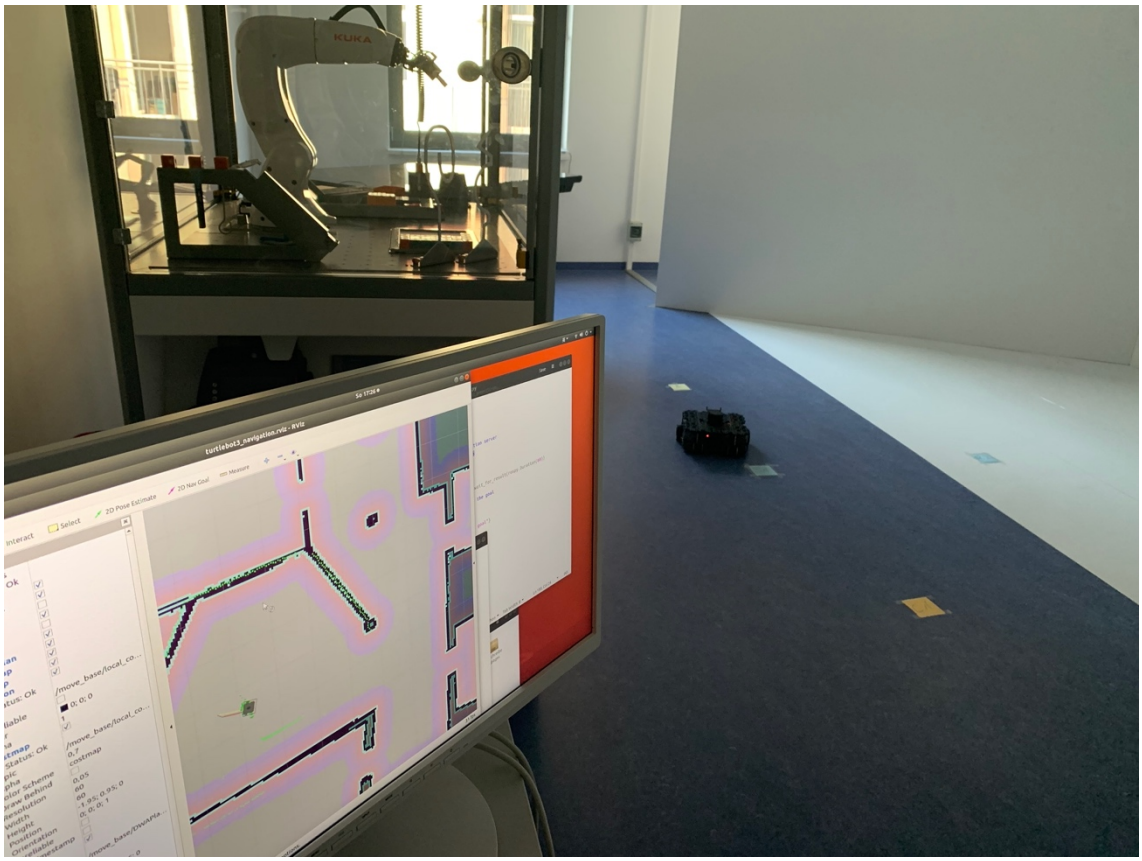

Abb. 5.5 Roboter fährt zum A1 Punkte
Quelle: Eigene Aufnahme in Institut für BauInformatik

Sobald der Punkt A1 erreicht ist, überprüfen der Roboter anhand des Bildes, ob es eine Fuge zwischen den beiden Dämmplatten gibt (siehe Abbildung 5.6) Wenn es eine Fuge gibt, wird die Mitte der beiden Platten durch einen grünen Strich markiert. Wenn es kein gibt, dann gibt es keine Fugen in der Mitte der Dämmplatten und die Platten sind korrekt verlegt, und Drücken Sie die Taste „esc“, um das Inspektionsfenster zu verlassen. Da es nur einen Zielpunkt A1 in der Kategorie A gibt, bedeutet dies, dass die anderen Zielpunkte nicht den gleichen Kategorien von Dämmplatten haben wie er. Es ist notwendig, zum Punkt N zurückzukehren, um das

Material abzuholen, siehe Abbildung 5.6.

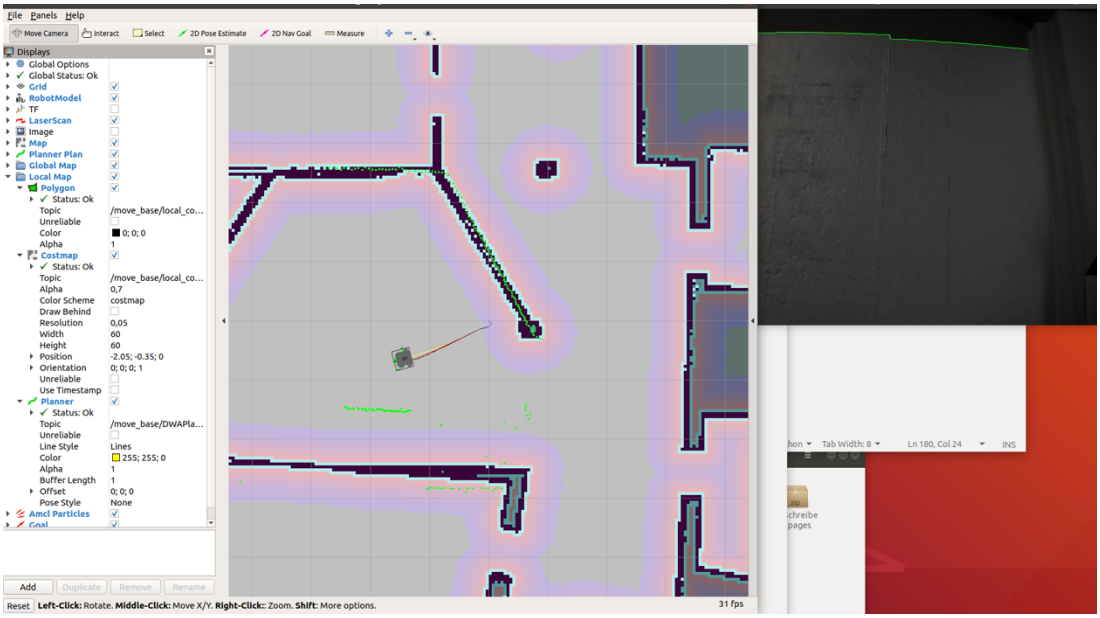

Abb. 5.6 Zurückfahren nach Überprüfung

Quelle: ScreenShot

Der Hauptteil des Arbeitsablaufs der Beispielimplementierung ist damit abgeschlossen, der Rest ist eine Wiederholung von 5.3.2. und 5.3.3. und wird daher hier nicht wiederholt. Die bei der Implementierung aufgetretenen Probleme werden in Abschnitt 5.4 analysiert und zusammengefasst.

5.4. Ergebnis Analyse

Der Roboter hat die Implementierung des Beispiels erfolgreich abgeschlossen und die Ergebnisse der Implementierung werden nun analysiert.

Der Roboter wurde in einer Simulationsumgebung, die auf der Grundlage des BIM-Modells entwickelt wurde, SLAM durchgeführt, und die in der Simulationsumgebung erhaltene Gitterkarte wird mit dem Move_Base-Knoten verwendet, um die im Programm angegebenen Zielpunkte anzufahren. Der ACML-Algorithmus wurde während des gesamten Laufs des Roboters verwendet, um das Koordinatensystem des Odom-Roboters jederzeit mit dem Koordinatensystem Map zu korrigieren, so dass während des gesamten Laufs des Roboters keine Drift des Odom-Koordinatensystems auftrat. Die Ergebnisse der Navigation waren dementsprechend zufriedenstellend. Abbildung 5.7 ist ein typisches Beispiel, bei dem jeder der sieben Koordinatenpunkte, in denen sich der Roboter bewegt, perfekt navigiert wird. Der Winkel des Roboters im Zielpunkt wird ebenfalls eingehalten.

Auf OpenCV wird hier nicht eingegangen, da es sich um dieselbe Methode handelt, die sowohl in der Beispielimplementierung als auch in der Simulation verwendet wird, und die bereits in den Simulationsergebnissen analysiert wurde.

Abb. 5.7 Roboter auf Zielpunkte
Quelle: Eigene Aufname in BauInformatik

Im Vergleich zu den Simulationsergebnissen in Kapitel 4 ist die Beispielimplementierung viel praktischer. In der Simulation fährt der Roboter immer genau zu den eingestellten Koordinaten, ohne dass die Transformationsbeziehung zwischen dem Koordinatensystem der Karte und dem Koordinatensystem des BIM-Modells berücksichtigt werden muss. Außerdem ist in der Simulationsumgebung alles fehlerfrei. Im Falle der Beispielimplementierung wäre der Prozess fehlgeschlagen, wenn der Roboter aufgrund eines Fehlers im LiDAR glaubte, mit einem Hindernis kollidiert zu sein, oder wenn der Roboter aufgrund eines plötzlichen Fehlers im Kilometerzähler oder in den Robotermotoren Informationen in Rrivz verloren hätte.

Wenn diese Technologie jedoch später auf realen Baustellen eingesetzt wird, müssen noch weitere Dinge berücksichtigt werden. Zum Beispiel, ob der Roboter kontinuierlich mit hoher Intensität arbeiten kann, und wie der Roboter angesichts seiner begrenzten Größe ein ausreichend genaues LiDAR tragen kann. Wenn die Baustelle sicherstellt, dass sich der Roboter immer in der gleichen

Netzwerkumgebung befindet wie der Computer, der ihn steuert. Dies sind alles Dinge, die in der Beispielimplementierung nicht gelöst werden können.

5.4. Künftige Usecase bei RopBau

Die TU Dresden entwickelt gemeinsam mit den Partnern Beas Technology aus Chemnitz und Dachdeckermeister Claus Dittrich GmbH aus Dresden im Projekt RopBau einen Roboter, der automatisch Dachdämmplatten verlegen kann. Die in dieser Arbeit verwendeten Methoden und Theorien können in diesem Projekt angewendet werden. Dasselbe kann durch die Entwicklung einer Simulationsumgebung aus dem BIM-Modell des Projekts und die Durchführung von SLAM in der Simulationsumgebung erreicht werden.

Bei der tatsächlichen Verlegung müssen die Koordinaten aus der globalen Costmap und dem entsprechenden CAD-Grundriss des Projekts umgewandelt werden, um zu berechnen, zu welchen Koordinaten der Roboter fahren muss, damit die Dachplatten verlegt werden können. Der gleiche Prozess kann bei der Visuellen Analyse zur Identifizierung und Überwachung von Dämmplatten eingesetzt werden.

Dabei ist zu beachten, dass es sich bei dem für das RopBau-Projekt entwickelten Roboter nicht um einen üblichen Roboterarm handelt, sondern um eine speziell für dieses Projekt entwickelte Greifvorrichtung. Er besitzt keine slebst Beweglichkeit, so dass die Verlegung der Dämmplatten ausschließlich durch den mobilen Teil des Roboters vorgenommen werden muss. Das bedeutet, dass trotz der Planung des Roboterwinkels während des Navigationsprozesses der tatsächliche Winkel des Roboters aufgrund von Fehlern nicht den Erwartungen entspricht, was zu Fehlern bei der Verlegung der Dachplatten führen kann. Dies ist ein Kernproblem, welches bei der weiteren Entwicklung des Projekts angegangen werden muss.

6 Zusammenfassung und Ausblick

In dieser Arbeit geht es um die Steigerung der Effizienz von Verlegung der Dämmungen für Dächer durch den Einsatz eines Logistikroboters mit SLAM. Durch die Verwendung von Turtlebot3 als Logistikroboter und die Entwicklung einer Simulationsumgebung wird Gmapping als SLAM-Algorithmus verwendet und vervollständigt den Arbeitsablauf des Roboters auf dem simulierten Dach. In dieser Arbeit wurde auch nachgewiesen, dass die von SLAM in der 1:1-Simulationsumgebung auf der Grundlage des BIM-Modells erstellten Karten in der realen Umgebung reibungslos verwendet werden können. Die folgenden Arbeiten wurden im Rahmen dieses Projekts durchgeführt.

Im Kapitel über die Methodik werden die spezifischen Methoden genannt, die in diesem Text verwendet werden. Die Analyse von Flachdächern zeigt, wie man Verlegepläne berechnet und aufstellt. Auch der Umsetzungsweg vom BIM-Modell zur Robotersimulationsumgebung wird geklärt. Aufgrund der Nichtverwendung des Ifc-Modells verfügt das Gebäudemodell in der Simulationsumgebung nur über geometrische Informationen. Dies ist jedoch für diese Arbeit nicht wichtig, da in dieser Arbeit nur die geometrischen Informationen der Gebäude für den Roboter in der Simulationsumgebung benötigt werden.

Für SLAM und Navigation werden die fundamentalen Prinzipien der wichtigsten verwendeten Algorithmen analysiert und in Zusammenhang mit ROS der Move_Base Knoten und die Kommunikation mit diesem Knoten. Aufgrund der Beschädigung der 3D-Kamera wurde der Aspekt der Computervision nicht vollständig entwickelt und nur der Canny-Operator wurde analysiert und zur Identifizierung der Objektkanten verwendet. Dieses Prinzip wurde auch für die Messung der Dachdämmplatten mit Hilfe einer Referenz verwendet. Es wurden auch Experimente durchgeführt, um die unterschiedlichen Schwellenwerte für verschiedene Dämmplattenmaterialien zu ermitteln und auf den Code anzuwenden.

Schließlich zeigen die Ergebnisse der Beispielimplementierung, dass der Einsatz von mobilen, autonomen Robotern mit SLAM-Technologie auf Baustellen im Inneren von Gebäuden auch Dächern theoretisch möglich ist. Durch SLAM in einer Simulationsumgebung und den anschließenden Einsatz des Roboters in einer realen Umgebung wird die Effizienz des Robotereinsatzes erheblich gesteigert.

Die Ergebnisse der Simulationen entsprachen völlig den Erwartungen der Arbeit und bestätigten, dass durch den Einsatz von SLAM-Robotern auf der Baustelle

tatsächlich erhebliche Effizienzsteigerungen erreicht werden können.

Der Teil der Computer Vision dieser Arbeit ist nicht völlig automatisiert, der Roboter kann nur erkennen, aber nicht automatisch das Ergebnis der Erkennung bestimmen, z.B. die Länge oder den Verlegestatus. Für den Betrieb ist nach wie vor ein Bediener erforderlich. Um eine maximale Automatisierung zu erreichen und die Effizienz der Bauarbeiten zu verbessern, ist es wichtig, dass der gesamte Arbeitsablauf autonom ist und menschliche Eingriffe so weit wie möglich vermieden werden. Daher kann in der zukünftigen Forschung maschinelles Lernen oder Deep Learning eingesetzt werden, um die Ergebnisse der visuellen Erkennung zusammenzufassen und automatisch zu erstellen.

VI Literaturverzeichnis

Monographien und Aufsätze in Fachzeitschriften

An, Juan; Mou, Hairong; Lu, Rui; Li, Youbing (2021): Localization and Navigation Analysis of Mobile Robot Based on SLAM. In: *J. Phys.: Conf. Ser.* 1827 (1), S. 12089. DOI: 10.1088/1742-6596/1827/1/012089.

Anil Mahtani; Luis Sanchez; Enrique Fernandez; Aaron Martinez (2016): Effective Robotics Programming with ROS - Third Edition. 3. Aufl. Birmingham: Packt Publishing Ltd.

Beetz, Jakob; Borrmann, André; Koch, Christian; König, Markus (Hg.) (2018): Building Information Modeling. Technology Foundations and Industry Practice. 1st ed. 2018. Cham: Springer International Publishing; Imprint: Springer.

Birkner, Britta; Willems, Wolfgang (2013): Flachdächer. In: Nabil A. Fouad (Hg.): Lehrbuch der Hochbaukonstruktionen. Wiesbaden: Springer Fachmedien Wiesbaden, S. 447–539.

Blankenbach, Jörg; Becker, Ralf (2020): BIM und die Digitalisierung im Bauwesen. In: Walter Frenz (Hg.): Handbuch Industrie 4.0: Recht, Technik, Gesellschaft. Berlin, Heidelberg: Springer, S. 777–795.

Borrmann, André; König, Markus; Koch, Christian; Beetz, Jakob (Hg.) (2021): Building Information Modeling. Technologische Grundlagen und industrielle Praxis. Springer Fachmedien Wiesbaden. 2., aktualisierte Auflage. Wiesbaden, Heidelberg: Springer Vieweg (VDI-Buch). Online verfügbar unter http://www.springer.com/.

Bradski, Gary R.; Kaehler, Adrian (2008): Learning OpenCV. Computer vision with the OpenCV library. 1. ed. Beijing, Köln: O'Reilly (Software that sees).

Fachregel für Abdichtungen Flachdachrichtlinie.

Gregor Fleischmann (2021): anzheitlicher Einsatz von BIG-OpenBIM bei der Ganzheitlicher Einsatz von BIG-Open-BIM bei der ÖBB-Infrastruktur AG als öffentlicher Auftraggeber. Lebenszyklusorientierter Einsatz der BIG-Open-BIM Methode im Bereich Eisenbahninfrastruktur. In: Christian Hofstadler und Christoph Motzko (Hg.): Agile Digitalisierung im Baubetrieb. Grundlagen, Innovationen, Disruptionen und Best Practices. Wiesbaden, Heidelberg: Springer Vieweg, S. 241–249.

Grisetti, G.; Stachniss, C.; Burgard, W. (2005): Improving Grid-based SLAM with Rao-

Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation. Proceedings of the 2005 IEEE International Conference on Robotics and Automation. Barcelona, Spain, 18.04.2005 - 22.04.2005: IEEE, S. 2432–2437.

Grisetti, Giorgio; Stachniss, Cyrill; Burgard, Wolfram (2007): Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. In: *IEEE Trans. Robot.* 23 (1), S. 34–46. DOI: 10.1109/TRO.2006.889486.

Gurel, Canberk (2018): REAL-TIME 2D AND 3D SLAM USING RTAB-MAP, GMAPPING, AND CARTOGRAPHER PACKAGES. University of Maryland. Maryland.

Hestermann, Ulf; Rongen, Ludwig (2018): Frick/Knöll Baukonstruktionslehre 2. 35. vollständig überarbeitete und aktualisierte Auflage. Wiesbaden: Springer Vieweg (Praxis).

Joseph, Lentin (2018): Mastering ROS for Robotics Programming, Second Edition. Design, build, and simulate complex robots using the Robot Operating System. Unter Mitarbeit von Jonathan Cacace. 2nd ed. Birmingham: Packt Publishing. Online verfügbar unter https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5314630.

Juan, Wu (Hg.) (2017): Proceedings of IEEE 13th International Conference on Electronic Measurement & Instruments. 2017 ICEMI : Oct. 20-22, 2017, Yangzhou, China. Institute of Electrical and Electronics Engineers. Piscataway, NJ: IEEE. Online verfügbar unter http://ieeexplore.ieee.org/servlet/opac?punumber=8258916.

Liu, Zhaohua; Cui, Zhi; Li, Yong; Wang, Wei (2020): Parameter Optimization Analysis of Gmapping Algorithm Based on Improved RBPF Particle Filter. In: *J. Phys.: Conf. Ser.* 1646 (1), Artikel 012004. DOI: 10.1088/1742-6596/1646/1/012004.

Murphy, Kevin; Russell, Stuart (2001): Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. In: Arnaud Doucet, Nando Freitas und Neil Gordon (Hg.): Sequential Monte Carlo Methods in Practice. New York, NY: Springer New York, S. 499–515.

Norzam, W.A.S; Hawari, H. F.; Kamarudin, K. (2019): Analysis of Mobile Robot Indoor Mapping using GMapping Based SLAM with Different Parameter. In: *IOP Conf. Ser.: Mater. Sci. Eng.* 705 (1), S. 12037. DOI: 10.1088/1757-899X/705/1/012037.

Pech, Anton; Hubner, Wolfgang; Zach, Franz (2011): Flachdach. [1. Aufl.]. Wien, New York: Springer (Baukonstruktionen, 9).

Pyo, YoonSeok; Cho, HanCheol; Jung, RyuWoon; Lim, TaeHoon (2017): ROS Robot Programming. 1. Aufl. Seoul: ROBOTIS Co.,Ltd.

Rong, Weibin; Li, Zhanjing; Zhang, Wei; Sun, Lining (2014): An improved Canny edge detection algorithm. In: 2014 IEEE International Conference on Mechatronics and Automation. 2014 IEEE International Conference on Mechatronics and Automation (ICMA). Tianjin, China, 03.08.2014 - 06.08.2014: IEEE, S. 577–582.

S.M.S. Elattar (2008): AUTOMATION AND ROBOTICS IN CONSTRUCTION: OPPORTUNITIES AND CHALLENGES. In: *Emirates Journal for Engineering Research* (13(2)), S. 21–26.

Saidi, Kamel S.; Bock, Thomas; Georgoulas, Christos (2016): Robotics in Construction. In: Springer Handbook of Robotics: Springer, Cham, S. 1493–1520. Online verfügbar unter https://link.springer.com/chapter/10.1007/978-3-319-32552-1_57.

Spengler, Arnim J.; Peter, Jacqueline (2020): Die Methode Building Information Modeling. Schnelleinstieg für Architekten und Bauingenieure. Wiesbaden, Heidelberg: Springer Vieweg (essentials). Online verfügbar unter http://www.springer.com/.

Stachniss, Cyrill; Leonard, John J.; Thrun, Sebastian (2016): Simultaneous Localization and Mapping. In: Springer Handbook of Robotics: Springer, Cham, S. 1153–1176. Online verfügbar unter https://link.springer.com/chapter/10.1007/978-3-319-32552-1_46.

Xu, Ziqi; Ji, Xiaoqiang; Wang, Meijiao; Sun, Xiaobing (2021): Edge detection algorithm of medical image based on Canny operator. In: *J. Phys.: Conf. Ser.* 1955 (1), S. 12080. DOI: 10.1088/1742-6596/1955/1/012080.

Zhang, Xuexi; Lai, Jiajun; Xu, Dongliang; Li, Huaijun; Fu, Minyue (2020): 2D Lidar-Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots. In: *Journal of Advanced Transportation* 2020, 1-14. DOI: 10.1155/2020/8867937.

Normen, Regelwerke und Richtlinien

DIN 18531-1	Abdichtung von Dächern sowie von Balkonen, Loggien und Laubengängen – Teil 1: Nicht genutzte und genutzte Dächer -Anforderungen, Planungs- und Ausführungsgrundsätze
Fachregel für Abdichtungen	Flachdachrichtlinie
DIN 4108-2	Wärmeschutz und Energie-Einsparung in Gebäuden – Teil 2: Mindestanforderungen an den Wärmeschutz
DIN 1986-100	Entwässerungsanlagen für Gebäude und Grundstücke – Teil 100: Bestimmungen in Verbindung mit DIN EN 752 und DIN EN 12056

Internet

Apel, J. (2022), Fachkräftemangel im Bau verschärft sich – Gewerkschaft schlägt Alarm, https://www.rnd.de/wirtschaft/baubranche-fachkraeftemangel-spitzt-sich-zu-ig-bau-schlaegt-alarm-QTEZG2Z6OJFARBP7JLJYLQEJX4.html, Stand: 25.08.2022

Beck, E.M. (2022), Digitalisierung in der Baubranche: Bereit für Bauen 4.0?, https://www.nevaris.com/blog/digitalisierung-in-der-baubranche/, Stand: 25.08.2022

Building Smart (2022), Industry Foundation Classes (IFC), https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/, Stand: 18.05.2022

Blender (2022), Blender, https://www.blender.org/, Stand: 15.06.2022

Cerciello, D. (2022), Warum das Flachdach aktuell wieder im Trend ist, https://www.myhomebook.de/projects/bauen/flachdach-hausbau-trend#h-flachdach-im-trend-die-vor-und-nachteile, Stand: 28. Juni 2022

HIRSCH Porozell (2022), Flachdach-Dämmplatte,

https://www.isobouw.de/produkte/flachdach/flachdach-daemmplatte.html, Stand: 18.07.2022

IfcOpenShell (2022), BlenderBIM ADD-ON, https://blenderbim.org/, Stand: 15.06.2022

Lumion- Act-3D (2022), Download Lumion LiveSync for Revit, https://support.lumion.com/hc/en-us/articles/360007538494-Download-Lumion-LiveSync-for-Revit, Stand: 15.06.2022

Meeussen, W. (2010), Coordinate Frames for Mobile Platforms, https://www.ros.org/reps/rep-0105.html, 25.09.2022

Open CV (2022a), Sobel Derivatives, https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html, Stand: 05.08.2022

Open CV (2022b), Canny Edge Detection, https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html, Stand: 05.08.2022

Open CV (2022c), Smoothing Images, https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html, Stand: 07.08.2022

OpenCV team (2022), Homepage, https://opencv.org/, Stand: 08.07.2022

Open Robotics (2013), Using the base controller with odometry and transform information, http://wiki.ros.org/pr2_controllers/Tutorials/Using%20the%20base%20controller%20with%20odometry%20and%20transform%20information, Stand: 25.06.2022

Open Robotics (2014), navfn, https://wiki.ros.org/navfn?distro=fuerte, Stand: 25.06.2022

Open Robotics (2017), Catkin Workspaces, http://wiki.ros.org/catkin/workspaces, Stand:25.06.2022

Open Robotics (2018a), costmap_2d, http://wiki.ros.org/costmap_2d, Stand:04.04.2022

Open Robotics (2018b), actionlib, http://wiki.ros.org/actionlib, Stand: 03.07.2022

Open Robotics (2018c), ROS Introduction, http://wiki.ros.org/ROS/Introduction,

Stand 03.04.2022

Open Robotics (2019a), ROS Tutorials MultipleMachines, http://wiki.ros.org/ROS/Tutorials/MultipleMachines, Stand:04.04.2022

Open Robotics (2019b), base_local_planner, https://wiki.ros.org/base_local_planner?distro=groovy, Stand: 25.06.2022

Open Robotics (2019c), actionlibDetailedDescription, http://wiki.ros.org/actionlib/DetailedDescription, Stand: 03.07.2022

Open Robotics (2020a), amcl, http://wiki.ros.org/amcl, Stand:04.04.2022

Open Robotics (2020b), move_base, http://wiki.ros.org/move_base, Stand:25.06.2022

Open Robotics (2020c), map_server, http://wiki.ros.org/map_server, Stand: 25.06.2022

Open Robotics (2020d), vision_opencv, http://wiki.ros.org/vision_opencv, Stand: 08.07.2022

Open Robotics (2021), Creating Package, http://wiki.ros.org/ROS/Tutorials/CreatingPackage, Stand:25.06.2022

Open Robotics (2022), Gazebo, https://gazebosim.org/home, Stand: 15.06.2022

OpenSLAM (2007), Gmapping, https://openslam-org.github.io/gmapping.html, Stand: 22.07.2022

Paul Bauder GmbH (2022), BauderPIRFA, https://www.bauder.de/de/flachdach/flachdach-produkte/waermedaemmung/flachdach/bauderpir-fa.html, Stand: 18.07.2022

Robotis (2022a), LDS-01, https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/, Stand: 01.04.2022

Robotis (2022b), Gazebo Simulation, https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation, Stand: 28.04.2022

ROCKWOOL GmbH (2022), Bitrock, https://www.rockwool.com/de/produkte/produktuebersicht/bitrock/,

Stand: 18.07.2022

ROS Answers (2012), Why navfn is using Dijkstra?, https://answers.ros.org/question/28366/why-navfn-is-using-dijkstra/, Stand: 25.06.2022

Van Ooyen, H. (2021), Roboter automatisieren die Bauindustrie, https://www.ingenieur.de/fachmedien/bauingenieur/special-digitalisierung/roboter-automatisieren-die-bauindustrie/, Stand: 25.08.2022

WCM.SYSTEMS (2018), Das Flachdach – Modischer Trend oder echte Alternative? https://dietrich-bedachung.de/2018/06/12/das-flachdach-modischer-trend-oder-echte-alternative/, Stand: 28. Juni 2022

VIIAnlagenverzeichnis

```
1   #!/usr/bin/env python
2   # coding:UTF-8
3   # Python 2/3 compatibility
4   from __future__ import print_function
5   import rospy
6   import actionlib
7   import tf
8   from actionlib_msgs.msg import *
9   from geometry_msgs.msg import Pose, Point, Quaternion, Twist
10  from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
11  from tf.transformations import quaternion_from_euler
12  from visualization_msgs.msg import Marker
13  import PyKDL
14  from math import radians, pi
15  import argparse
16  import imutils
17  import cv2
18  import cv2 as cv
19  import numpy as np
20  from imutils import perspective
21  from scipy.spatial import distance as dist
22  from imutils import contours
23
24  # built-in module
25  import sys
26
27  #Initialisierung zwei Zähler m und n
28  m = 0
29  n = 0
30
31
32  class Verlegung():
33      def __init__(self):
34          rospy.init_node('go', anonymous=False)
35
36          rospy.on_shutdown(self.shutdown)
37
38
39          # Erstellen einer Liste mit den Ziel-Quaternionen (Ausrichtungen)
```

```
40          quaternions = list()
41
42          # Zuerst werden die Eckausrichtungen als Euler-Winkel definiert
43          euler_angles = (pi/2, pi, 3*pi/2, 0)
44
45          # Dann die Winkel in Quaternionen umwandeln
46          for angle in euler_angles:
47              q_angle = quaternion_from_euler(0, 0, angle, axes='sxyz')
48              q = Quaternion(*q_angle)
49              quaternions.append(q)
50
51          # Eine Liste erstellen, die die Wegpunktpositionen enthält
52          waypoints = list()
53
54          # Fügen der Liste jeden der vier Wegpunkte hinzu.  Jeder Wegpunkt
55          waypoints.append(Pose(Point(0.00, 0.00, 0.0), quaternions[2]))
56          waypoints.append(Pose(Point(-0.30, 3.50, 0.0), quaternions[0]))
57          waypoints.append(Pose(Point(0.00, 0.0, 0.0), quaternions[2]))
58          waypoints.append(Pose(Point(-0.30, 2.50, 0.0), quaternions[0]))
59          waypoints.append(Pose(Point(0.70, 2.50, 0.0), quaternions[0]))
60          waypoints.append(Pose(Point(0.00, 0.0, 0.0), quaternions[2]))
61          waypoints.append(Pose(Point(-0.30, 1.5, 0.0), quaternions[0]))
62          waypoints.append(Pose(Point(0.70, 1.5, 0.0), quaternions[0]))
63          waypoints.append(Pose(Point(1.70, 1.5, 0.0), quaternions[3]))
64
65          # Herausgeber zur manuellen Steuerung des Roboters (z. B. um ihn anzuhalt
    en)
66          self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
67
68          # Abonnieren des move_base-Action Client
69          self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction
    )
70
71          rospy.loginfo("Waiting for move_base action server...")
72
73          # 60 Sekunden warten, bis der Aktionsserver verfügbar ist
74          self.move_base.wait_for_server(rospy.Duration(60))
75
76          rospy.loginfo("Connected to move base server")
77          rospy.loginfo("Starting navigation test")
78
79          # Initialisierung eines Zählers zur Verfolgung von Wegpunkten
```

```
80          i = 0
81
82          # Cycle through the four waypoints
83          while i < 10 and not rospy.is_shutdown():
84
85              # Intialisieren des Wegpunktziels
86              goal = MoveBaseGoal()
87
88              # Verwenden des Map-Frame zur Definition von Zielposen
89              goal.target_pose.header.frame_id = 'map'
90
91              # Den Zeitstempel auf "jetzt" setzen
92              goal.target_pose.header.stamp = rospy.Time.now()
93
94              # Setzen der Zielpose auf den i-ten Wegpunkt
95              goal.target_pose.pose = waypoints[i]
96
97              # Starten des Roboters in Richtung des Ziels
98              self.move(goal)
99
100             #Verwendung von Listen zur Positionsbestimmung
101             i += 1
102             Bilden = [1,3,6,10,17]
103
104             Bilden = set(Bilden)
105             fn1 = str(i)
106             fn2 = '.jpg'
107             fn  = fn1 + fn2
108
109             #Auswahl, ob Messung oder Prüfung
110             if i in Bilden:
111                 self.identifizieren(fn)
112             else:
113                 self.prufen(fn)
114
115     def move(self, goal):
116
117             # Senden der Zielposition an den MoveBaseAction-Server
118             self.move_base.send_goal(goal)
119
120             # 1 Minute Zeit für die Anfahrt
```

```
121             finished_within_time = self.move_base.wait_for_result(rospy.Duration(
    60))
122 
123             # Falls nicht rechtzeitig ankommen, das Ziel abbrechen
124             if not finished_within_time:
125                 self.move_base.cancel_goal()
126                 rospy.loginfo("Timed out achieving goal")
127             else:
128                 # Erreichen des Zielpunkts
129                 state = self.move_base.get_state()
130                 if state == GoalStatus.SUCCEEDED:
131                     rospy.loginfo("Goal succeeded!")
132                     rospy.sleep(1)
133 
134     def prufen(self,fn):
135            # Definieren einer Binärliste zur Speicherung der Schwellenwerte
136             index = [[1772,3773],
137                      [1349,3436],
138                      [1643,3085],
139                      [1593,2518],
140                          [1413,4000],
141                      [1126,3852]]
142             # Bilder lesen
143             img = cv.imread(cv.samples.findFile(fn))
144             cv.namedWindow('Dammung')
145 
146             global m
147 
148            # Dedizierte Graustufenkarte und entsprechende Schwellenwerte
149             gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
150             thrs1 = index[m][0]
151             thrs2 = index[m][1]
152 
153            # Kantenerkennung mit dem Canny-Operator
154             edge = cv.Canny(gray, thrs1, thrs2, apertureSize=5)
155             vis = img.copy()
156             vis = np.uint8(vis/2.)
157             vis[edge != 0] = (0, 255, 0)
158            # Bilder anzeigen
159             cv.imshow('Dammung', vis)
160             ch = cv.waitKey(5)
161 
```

```
162             if ch == 27:
163                     break
164             m += 1
165             print('Done')
166
167     def identifizieren(self,fn):
168             # Definieren einer Funktion zur Ermittlung des Mittelpunkts
169             def mittel(A, B):
170                 return ((A[0] + B[0]) * 0.5, (A[1] + B[1]) * 0.5)
171
172             # Festlegen der Parameter der Referenz
173             ap = argparse.ArgumentParser()
174             ap.add_argument("--width", type=float)
175             args = vars(ap.parse_args())
176
177             # Auswahl von Gauß-
    Filtern unterschiedlicher Größe für verschiedene Bilder
178             gau1 = [5,5,3]
179             global n
180
181             # Bilder lesen,  und in Graustufen umgewandelt
182             image = cv.imread(cv.samples.findFile(fn))
183             gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
184
185             #Gauß-Filterung
186             gray = cv2.GaussianBlur(gray, (gau1[n], gau1[n]), 0)
187
188             # Erkennung von Kanten
189             edged = cv2.Canny(gray, 20, 80)
190             edged = cv2.dilate(edged, None, iterations=1)
191             edged = cv2.erode(edged, None, iterations=1)
192
193             # Objekte finden
194             cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_AP
    PROX_SIMPLE)
195             cnts = imutils.grab_contours(cnts)
196
197             # Sortierung der Umrisse von links nach rechts
198             (cnts, _) = contours.sort_contours(cnts)
199             pixelsPerMetric = None
200
201             n += 1
```

```
202
203             # Berechnen der Breite des Objekts entsprechend der Größe des Umrisse
    s und Mittelwertbildung
204             for c in cnts:
205                 if cv2.contourArea(c) < 1000:
206                     continue
207
208             orig= image.copy()
209                 box = cv2.minAreaRect(c)
210                 box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoint
    s(box)
211                 box = np.array(box, dtype="int")
212
213                 box = perspective.order_points(box)
214                 cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2)
215
216                 for (x, y) in box:
217                     cv2.circle(orig, (int(x), int(y)), 5, (0, 0, 255), -1)
218
219                 (lo, ro, ru, lu) = box
220
221                 (ObenX, ObenY) = mittel(lo, ro)
222                 (UnterX, UnterY) = mittel(lu, ru)
223
224                 cv2.circle(orig, (int(ObenX), int(ObenY)), 5, (255, 0, 0), -1)
225                 cv2.circle(orig, (int(UnterX), int(UnterY)), 5, (255, 0, 0), -
    1)
226
227                 cv2.line(orig, (int(ObenX), int(ObenY)), (int(UnterX), int(UnterY
    )), (255, 0, 255), 2)
228
229                 hohe = (dist.euclidean((ObenX, ObenY), (UnterX, UnterY))+dist.euc
    lidean(lo, lu)+dist.euclidean(ro, ru))/3
230
231                 if pixelsPerMetric is None:
232                     pixelsPerMetric = hohe / args["width"]
233
234                 lange = hohe / pixelsPerMetric
235
236                 cv2.putText(orig, "{:.1f}cm".format(lange),
237                     (int(ObenX ), int(ObenY+10)), cv2.FONT_HERSHEY_SIMPLEX,
238                     0.65, (255, 255, 255), 2)
```

```
239                     # Bilder anzeigen
240                 cv2.imshow("Image", orig)
241                 cv2.waitKey(0)
242 
243     def shutdown(self):
244         rospy.loginfo("Stopping the robot...")
245 
246         # Alle aktiven Ziele stornieren
247         self.move_base.cancel_goal()
248         rospy.sleep(2)
249 
250         # Anhalten des Roboters
251         self.cmd_vel_pub.publish(Twist())
252         rospy.sleep(1)
253 
254 if __name__ == '__main__':
255     try:
256         Verlegung()
257     except rospy.ROSInterruptException:
258         rospy.loginfo("Verlegung fertig.")
259 
```

VIII Thesen zur Diplomarbeit

1. Intelligente Roboter können bei Bauprojekten die menschliche Arbeitskraft teilweise ersetzen und die Baueffizienz steigern.

2. Der weit verbreitete Einsatz von BIM-Modellen in der Bauplanung bietet große Möglichkeiten und Entwicklungspotenzial für den Einsatz von Robotern.

3. SLAM eignet sich hervorragend für den Einsatz von Robotern im Bauwesen

4. Damit Roboter oder neue Technologien, wie z. B. Computer Vision, im Bauwesen eingesetzt werden können, müssen sie an die Baumaterialien oder die entsprechenden Industrienormen angepasst werden.

5. Es besteht weiterhin Bedarf an der Entwicklung und Forschung von Robotern, die hochgradig autonom sind und unabhängig vom Menschen arbeiten können.

6. Roboter auf Baustellen müssen mit sehr komplexen und dynamischen Arbeitsumgebungen zurechtkommen. Um dies zu bewältigen, sind die Roboter mit leistungsfähigen Prozessoren und Sensoren sowie mit Algorithmen ausgestattet, die für komplexe Situationen optimiert sind.