



INSTITUT FÜR INFORMATIK  
AG TECHNISCHE INFORMATIK

*Masterarbeit*

# Loop Closure in TSDF basiertem SLAM

Patrick Hoffmann

November 2022

Erstgutachter: Prof. Dr. Mario Porrmann  
Zweitgutachter: Alexander Mock, M. Sc.



## Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Implementierung und Konzeption einer Lösung zur Detektion von **Schleifenschlüssen** in einem auf **Truncated Signed Distance Field (TSDF)** basierenden **Simultaneaous Localization and Mapping (SLAM)** Ansatz und der nachfolgenden Optimierung der Robotertrajektorie und TSDF-Karte. Zur Optimierung der Trajektorie des Roboters nach der Identifikation eines Schleifenschlusses kommt die Bibliothek **GTSAM** zum Einsatz. Es wird evaluiert ob und unter welchen Voraussetzungen eine Nachbearbeitung auf Basis einer fertigen TSDF Karte mit zugehöriger initialer Trajektorie möglich ist und zusätzlich der Einsatz in einem inkrementellen SLAM Ansatz geprüft. Darüber wird untersucht, inwiefern ein Teilupdate der TSDF basierten Karte möglich ist.

## Abstract

This paper deals with the implementation and design of a solution for the detection of **loop closures** in a **Truncated Signed Distance Field (TSDF)** based **Simultaneaous Localization and Mapping (SLAM)** approach and the subsequent optimization of the robot trajectory and TSDF map. The **GTSAM** library is used to optimize the robot's trajectory after identifying a loop closure. It is evaluated whether and under which conditions a postprocessing based on a finished TSDF map with associated initial trajectory is possible and additionally the use in an incremental SLAM approach is examined. Furthermore, it will be investigated to what extent a partial update of the TSDF based map is possible.



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Herangehensweise . . . . .	3
1.2.1 Fallback . . . . .	3
<b>2 Stand der Forschung</b>	<b>5</b>
<b>3 Grundlagen</b>	<b>7</b>
3.1 Mathematische Grundlagen . . . . .	7
3.1.1 Konventionen . . . . .	7
3.1.2 Koordinatensysteme . . . . .	7
3.2 TSDF . . . . .	11
3.2.1 TSDF-Karte . . . . .	13
3.2.2 TSDF-Update . . . . .	13
3.3 SLAM . . . . .	13
3.4 Loop Closure . . . . .	15
<b>4 Datenassoziationen</b>	<b>17</b>
4.1 Ansatz . . . . .	17
4.2 Serialisierung . . . . .	18
4.3 Algorithmen . . . . .	18
4.3.1 Ray-Tracing . . . . .	18
4.3.2 Bresenham . . . . .	23
4.3.3 Ergebnisse . . . . .	26
4.3.4 Evaluation . . . . .	27
4.4 Loop Closure . . . . .	28
4.5 Kartenupdate . . . . .	31
4.6 Evaluation . . . . .	33
<b>5 Schleifenschlüsse</b>	<b>35</b>
5.1 Detektion von Schleifenschlüssen . . . . .	35
5.2 Optimierung des Pose-Graphen . . . . .	41
5.3 Evaluation der Graph-Optimierung . . . . .	42

5.3.1	Datensatz: Hannover-1 . . . . .	42
5.3.2	Datensatz: Chemnitzer Oper . . . . .	46
5.3.3	Datensatz: Campus Universität Osnabrück . . . . .	48
<b>6</b>	<b>Karten-Update Strategien</b>	<b>51</b>
6.1	Globales Update der Karte . . . . .	52
6.2	Partielles Update der Karte . . . . .	53
6.3	Evaluation der Update-Strategien . . . . .	56
<b>7</b>	<b>Zusammenfassung und Ausblick</b>	<b>59</b>
7.1	Zusammenfassung . . . . .	59
7.2	Ausblick . . . . .	59
7.2.1	Optimierungen . . . . .	59
<b>A</b>	<b>Anhang</b>	<b>61</b>

# Kapitel 1

## Einleitung

Diese Arbeit beschäftigt sich mit der Konzeption und Implementation einer Lösung zum *Loop Closure (Schleifenschluss)* Problem in *Truncated Signed Distance Fields* (TSDF) basiertem *Simultaneous Localization and Mapping* (SLAM). Diese Arbeit setzt auf den Konzepten von Eisoldt et al. [10] auf, die einen TSDF basierten, hardware-beschleunigten SLAM Ansatz (*HATSDF-SLAM*) entwickelt und dessen Effizienz und Funktionalität verifiziert haben.

Unter SLAM versteht man den Prozess der Erstellung einer Karte einer unbekannten Umgebung, bei gleichzeitiger Positionsbestimmung innerhalb der erstellten Karte. Das SLAM Problem ist ein *Henne-Ei-Problem*, da eine vollständige Karte benötigt wird, um die Pose eines Roboters akkurat bestimmen zu können, während auf der anderen Seite eine akkurate Pose benötigt wird, um eine vollständige Karte aufbauen zu können. Für die Lösung dieses Problems gibt es sowohl in 2D als auch 3D verschiedene Lösungsansätze, die in Kapitel 2 grob eruiert werden. Diese Arbeit beschränkt sich auf 6D SLAM. Unter einem Schleifenschluss versteht man in der mobilen Robotik das Problem, geschlossene Kreise in abgelaufen Pfaden durch die im SLAM-Prozess erschlossene, zunächst unbekannte Umgebung, zu identifizieren. Nach der Identifikation einer Schleife zwischen zwei Posen  $P_i$  und  $P_j$  wird die approximative Transformation  $T_{i \rightarrow j}$  bestimmt und mit der Transformation  $T_{i \rightarrow j}^{graph}$  verglichen, die aus derzeitigen Posegraphen hervorgeht. Die Differenz zwischen diesen Transformation  $T_{error}$  wird näherungsweise als der Fehler angesehen, der sich beim SLAM zwischen den Posen  $P_i$  und  $P_j$  akkumuliert hat. In einem Optimierungsschritt wird der Fehler  $T_{error}$  durch eine Anpassung des Pose-Teilgraphen zwischen  $P_i$  und  $P_j$  kompensiert. Dies sorgt für eine konsistenteren Pfad und damit verbunden für eine akkurate Karte. Als Basis für die Lösung dieses Problems dient der Ansatz von Lu und Milios [16], die ein Netzwerk aus 2D-Poserelationen erstellen, das sowohl Relationen enthalte, die aus dem Scan-Matching abstammen, als auch Relationen aus Odometrie Messungen enthalte [16]. Borrmann et al. [4] erweitern den Ansatz von [16] auf drei Dimensionen und erstellen einen Pose-Graphen zur Speicherung von Pose-Relationen. Zur Registrierung aufeinanderfolgender Datenscans verwenden Borrmann et al. [4] den *Pairwise Iterative Closest Point Algorithm* (Pairwise ICP). Zur Detektion eines Schleifenschlusses wird eine einfache Distanz-Heuristik verwendet. Shang et al. [23] beschreiben eine vergleichbare Heuristik zur Identifikation von Schleifenschlüssen. Der gewählte Ansatz

verwende eine naive, aber effektive Heuristik, die auf der euklidischen Distanz zwischen den Posen des Graphen basiert. Posen, deren euklidische Distanz zur aktuell betrachteten Pose geringer ist als eine festgelegte Schwelle, werden als Kandidaten angesehen. Jeder dieser Kandidaten werde über ein Scan-Matching der zugehörigen Punktwolken und der Evaluation des Ergebnisses des Scan-Matching validiert [23]. [23] verwendet einen Faktor-Graphen zur Speicherung der Pose-Relationen. Hierfür wird die Implementation des Faktor-Graphen von **GTSAM** [9] verwendet, der ebenfalls in dieser Arbeit zur Speicherung und Optimierung der Pose-Relationen verwendet wird. Wichtige Voraussetzung für die Optimierung der Posen im Graphen ist eine Assoziation der Posen mit den zugehörigen Umgebungsdaten. Im Falle von Borrman et al. [4] sind dies jeweils die zum Zeitpunkt aufgenommenen Punktwolken. [23] sorgt durch die Selektion von **Key-Frames** für einen spärlich besetzten Faktor-Graphen und sorge so für eine Ausgeglichenheit zwischen Speicherbedarf und der Dichte der Karte. Die Menge der Key-Frames ist eine Untermenge der gesamten Lidar-Frames die bei der Kartierung aufgenommen wurden. Ein Frame wird als Key-Frame genutzt, wenn zum zuletzt gewählten Key-Frame eine festgelegte Schwelle für die Translation oder Rotation überschritten wird. Diese Assoziation ist bei TSDF basiertem-SLAM, das eine diskretisierte TSDF-Karte verwendet, nicht explizit möglich, solange nicht zusätzlich die entsprechenden Punktdaten abgespeichert werden. Wird ein Schleifenschluss detektiert, muss dann die gesamte Karte auf Basis der abgespeicherten Punktdaten neu erstellt werden. Dies ist speicher- und zeitaufwändig. Dieses Problem markiert den Hauptbeitrag dieser Arbeit. Es gilt zu untersuchen, welcher Teil der TSDF Karte mit welcher Pose im Graph assoziiert werden kann, um anschließend die bereits existierende Karte zu modifizieren und zu optimieren. Dieses Problem wird im Folgenden untersucht und verschiedene Herangehensweisen werden eruiert.

## 1.1 Motivation

Ziel dieser Arbeit ist die Lösung des Schleifenschluss Problems für TSDF basierte SLAM Verfahren. Als Basis dient der Ansatz von Eisoldt et al. [10], der - obgleich performant und funktional - wie die meisten SLAM Verfahren bei der Kartierung großer Umgebungen stark anfällig für Drift ist. Um diesem Problem entgegen zu wirken, soll durch die Integration von Schleifenschlüssen in TSDF-Karten die Kartierung größerer Umgebungen ermöglicht werden. Um dieses Ziel zu erreichen, soll in einem ersten Ansatz untersucht werden, ob eine bereits generierte Karte mit bekannter Posehistorie durch die Integration von Schleifenschlüssen zur Optimierung der Trajektorie verbessert werden kann. Es gilt zu untersuchen, welche Voraussetzungen für eine solche Optimierung gegeben sein müssen und ob diese Voraussetzungen in diesem ersten Szenario erfüllt werden können. Auf Basis dieser Untersuchung soll eine Einschätzung zur Nutzung von Schleifenschlüssen in einem TSDF basierten SLAM Ansatz als Nachbehandlungs-/Post-Processing-Schritt geben werden. Diese Einschätzung markiert einen wesentlichen Meilenstein dieser Arbeit, bei dem entschieden wird das beschriebene Szenario weiter zu verfolgen, oder eine Integration in einen SLAM Ansatz anzustreben und zur Laufzeit Optimierungen an der Trajektorie und Karte basierend auf identifizierten Schleifenschlüssen vorzunehmen.

Grundlegende Voraussetzung für die Optimierung von Pfad und Karte ist eine Assoziation zwischen den einzelnen Posen des Pfades mit den jeweiligen zugehörigen Umgebungsdaten. Erstes Ziel des genannten ersten Szenarios ist dementsprechend die Ermittlung von Datenassoziationen zwischen

den Posen und den zugehörigen Teilen der diskretisierten TSDF-Karte. Diese Assoziationen dienen als Ersatz zu den von von Borrmann et al. [4] und Shang et al. [23] genutzten vorgefilterten Punktwolken beziehungsweise **Key-Frames**, die mit den zugehörigen Posen assoziiert werden. In einer ersten Implementation soll dazu mittels Ray-Tracing eines simulierten Laserscans und der Detektion von Schnittpunkten mit der TSDF-Karte eine passende Indizierung und Zuordnung der Zellen ermöglicht werden. Kapitel 4 erörtert und evaluiert diesen Ansatz, beschreibt mögliche Probleme und Fallstricke und definiert, wie im weiteren Verlauf der Arbeit vorgegangen wird.

Diese Arbeit soll als Proof-of-Concept für weitere Arbeiten auf diesem Gebiet dienen und Herausforderungen und Fallstricke aufzeigen. Es soll geklärt werden wie und auf welche Weise Schleifenschlüsse in einen TSDF basierten SLAM-Ansatz integriert werden können, ob eine Nachbehandlung möglich ist, sowie ob und wie ein partielles Update der Karte möglich ist. In einem Ausblick gilt es zu analysieren, ob die Implementierungen sinnvoll beschleunigt werden können, um sie in ein Live Karierungssystem, wie HATSDF-Slam einzubauen zu können, um zur Laufzeit Schleifen zu erkennen und die Karte zu optimieren. Die Implementation dieses Prototyps wird ausschließlich in Software realisiert, allerdings erfolgt eine Untersuchung des Software-Prototyps auf Potenziale zur Hardware-Beschleunigung um Raum für Optimierungen im Rahmen zukünftiger Arbeiten zu eröffnen.

Nachfolgende Sektion nimmt erneut Bezug auf den gegebenen Ansatz, stellt die Offenheit des Themas heraus und definiert mögliche Fallbacks zum Post-Processing Szenario auf Basis der ermittelten Datenassoziationen.

## 1.2 Herangehensweise

Wie bereits in der vorigen Sektion definiert, ist das genaue Ziel dieser Arbeit offen, da die Ermittlung von Datenassoziationen aus einer generierten TSDF Map, um diese für Schleifenschlüsse zu verwenden, ein rein experimenteller Ansatz ist. Hier gilt es herauszustellen ob dieser Ansatz erfolgreich ist oder - im anderen Fall - Hürden und Probleme aufzuzeigen, sowie mögliche Lösungsansätze für zukünftige Arbeiten zu skizzieren, die nicht in den Zeitrahmen dieser Arbeit passen. Das grundlegende Ziel der Integration von Schleifenschlüssen allerdings bleibt. Zunächst werden in Kapitel ?? Lösungsansätze dokumentiert, mit deren Hilfe Datenassoziationen zwischen der TSDF-Karte und der Posehistorie generiert werden können. Dies wird evaluiert und die weitere Vorgehensweise diskutiert.

Folgender Abschnitt definiert mögliche Fallbacks zur Generation von Datenassoziationen aus dem TSDF und alternative Möglichkeiten zum (partiellen) Update der TSDF Karte.

### 1.2.1 Fallback

Sollte sich im Laufe dieser Arbeit herausstellen, dass die Generation von Datenassoziationen und das Update der TSDF Karte auf Basis dieser Assoziationen entweder nicht möglich ist oder Lösungsansätze nicht mit dem zeitlichen Rahmen dieser Arbeit vereinbar sind, sollen alternative Möglichkeiten zum Update der Karte definiert und entwickelt werden. Zunächst wird dabei ein globales Update der gesamten TSDF Karte auf Basis der zu den Posen gehörigen Punktwolken

angestrebt. Dieses soll zusätzlich um ein partielles Update der betroffenen Kartenbereiche erweitert werden. In diesem Fall wird in einem Ausblick zusätzlich Bezug zu zukünftigen Arbeiten und Lösungsmöglichkeiten für die Generation und Nutzung von TSDF-Pose-Datenassoziationen genommen werden.

## Kapitel 2

# Stand der Forschung

*Simultaneous Localization and Mapping (SLAM)* ist eines der größten Forschungsgebiete in der mobilen Robotik. Die Forschungen zu diesem Problem reichen zurück bis vor die Jahrtausendwende. Hauptbestandteil vieler SLAM Ansätze sind 3D Punktwolken als Repräsentation der Umgebung, die mit Sensoren wie Laserscannern und Tiefenbildkameras aufgenommen werden können. Dabei werden von verschiedenen Positionen im dreidimensionalen Raum (**Posen**) Punktwolken aufgenommen. Um diese Punktwolken nahtlos aneinander anknüpfen zu können, müssen diese miteinander **registriert** werden. Die Registrierung bestimmt die 3D Transformation (siehe dazu Kapitel 3.1.2) zwischen zwei Punktwolken, wobei die **Scan-Punktwolke** an die **Model-Punktwolke** registriert wird. Einer der bekanntesten Lösungen zur Registrierung dreidimensionaler Punktdaten ist der **Iterative Closest Points (ICP)** Algorithmus nach Besl & McKay [3], der in einer Abwandlung ebenfalls von Borrmann et al. im GraphSLAM [4] verwendet wird. Besl & McKay [3] bestimmen die gesuchte 3D Transformation zwischen zwei Punktwolken durch eine Minimierung der Summe der quadrierten Distanzen der nächsten Punkte zwischen den beiden Punktwolken durch eine **Singular Value Decomposition (SVD)**. Die Konvergenz des ICP Algorithmus ist stark gekoppelt an die initiale Schätzung zwischen den beiden Laserscans. Ist diese nicht gut gewählt, konvergiert ICP häufig in lokale Minima, was zu ungewünschten Ergebnissen führt [12]. Bis heute gibt es zahlreiche Variationen und Verbesserungen des ICP Algorithmus. Chen & Medioni [6] erweitern ICP durch die Nutzung von Oberflächen-Normalen des Models und erweitern so die Kostenfunktion von ICP. Sie minimiert nun die Summe der quadrierten Distanzen zwischen einem Scarpunkt und der durch den Modelpunkt und die Oberflächennormale am Modelpunkt beschriebenen Ebene. Dieser Algorithmus wird auch **Normal ICP (NICP)** genannt. Nach [12] reduziert NICP die Anzahl Iterationen und konvergiert schneller gegen eine gewisse Grenze als ICP. Segal et al. [22] erweitern die Ideen aus [6] um eine **Plane-to-Plane** Metrik. Neben genannten Methoden existieren zahlreiche weitere Variationen von ICP, die jeweils kleine Anpassungen vornehmen wie Chetverikov et al. [7], die einen **Least Trimmed Squares** Ansatz zur Fehlerminimierung verwenden und so auch für Punktwolken anwendbar ist, die sich weniger als 50% überlappen.

Erste Forschungen mit TSDF basiertem SLAM stammen aus dem letzten Jahrzehnt. Izadi

et al. [13] nutzen ein TSDF-Voxelgrid und eine Kinect-Tiefenkamera, um Umgebungen zu kartieren, während ein Nutzer die Kinect Kamera durch die Umgebung schwenkt. Whelan et al. [26] optimieren diesen Ansatz durch Nutzung eines Ringbuffers zur Kartierung großer Umgebungen. Im Gegensatz zu genannten TSDF Verfahren, nutzen Eisoldt et al. [10] einen Hardware beschleunigten TSDF basierten SLAM Ansatz, sowie einen 3D-Laserscanner anstelle einer Tiefenkamera. Eisoldt et. al [10] nutzen zur Registrierung neuer Punktwolken an die TSDF Karte einen **Point-to-TSDF** Ansatz, der die Distanzen von Punkten zur durch die TSDF implizit beschriebenen Oberfläche entlang des Gradienten innerhalb der TSDF minimiert.

Borrmann et al. [4] schufen in ihrer Arbeit eine gute Grundlage für die Integration von Schleifenschlüssen in SLAM-Verfahren auf Basis von Pose-Graphen, die bis heute vielfach verwendet wird. Darauf aufbauend optimieren Sprickerhof et al. [24] den Schleifenschluss-Ansatz durch Nutzung einer heuristischen Schleifenschluss-Technik, die im Gegensatz zu bisherigen Methoden einen dünn besetzten SLAM-Graphen nutzen. McCormac et al. [18] schlagen ein Online-SLAM System, welches eine dauerhafte und genaue 3D Karte beliebiger rekonstruierter Objekte darstellt [18]. Die Rekonstruktionen sind als TSDF realisiert. [18] schlägt ebenfalls eine Integration von Schleifenschlüssen vor, diese verändert allerdings lediglich die relativen Poseschätzungen, aber führt zu keiner Verformung der durch die TSDF beschriebenen Objekte. An dieser Stelle wird also kein Update der TSDF durch Schleifenschlüsse vorgenommen. Prisacariu et al. [19] unterteilen den 3D-Raum in starre TSDF-Teilkarten und optimieren die relativen Positionen zwischen diesen. Nach [19] ist eine anschließende Generation der globalen Karte durch ein Zusammenführen der Teilkarten möglich. [19] verwendet für jede Pose des Graphen eine eigene Teilkarte, die in sich konsistent ist. [19] erreichen eine globale Konsistenz durch eine Graph-Optimierung, die zusätzlich Schleifenschlüsse berücksichtigt. Optimiert werden die zu den Teilkarten zugehörigen Posen. Die TSDF-Teilkarten selbst werden durch die Graph-Optimierung nicht angepasst. Ähnlich wie [4] nutzen Shan et al. [23] in ihrem Feature basierten SLAM Ansatz die euklidische Distanz zur Bestimmung von Kandidaten für Schleifenschlüsse, die durch eine Evaluation des **Fitness-Scores** von ICP nach Registrierung der zugehörigen Punktwolken der beiden Posen der jeweiligen Kandidaten verifiziert werden.

An dieser Stelle setzt diese Arbeit nun an und integriert auf Basis der zuvor genannten Ansätze, Schleifenschlüsse in einen TSDF basierten SLAM Ansatz und die TSDF Karte entsprechend der Änderungen an der Roboter-Trajektorie zu optimieren.

# Kapitel 3

## Grundlagen

### 3.1 Mathematische Grundlagen

Diese Sektion beschreibt die wesentlichen mathematischen Konzepte, die in dieser Arbeit zur Verwendung kommen.

Kapitel wurde bisher nicht weiter angepasst, Änderungen von Alex müssen noch eingearbeitet werden, Fokus lag auf Stand der Forschung, Map update

#### 3.1.1 Konventionen

Transformationen werden im Folgenden folgendermaßen betitelt:

Konventionen einfügen

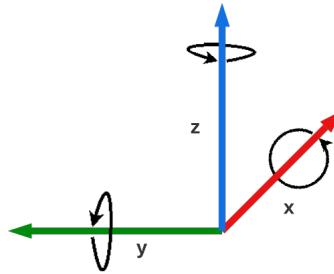
#### Pfad

#### 3.1.2 Koordinatensysteme

Ein wesentliches Konzept bei der Verarbeitung räumlicher Daten ist die Verwendung von Koordinatensystemen. Sie werden genutzt um die Positionen von Daten und Objekten im Raum zu beschreiben. Koordinatensysteme können für sich alleine stehen oder relativ zu anderen Koordinatensystemen. Im Dreidimensionalen besitzt ein Koordinatensystem drei verschiedene Achsen (x, y und z-Achse), die jeweils im 90 Grad Winkel zueinander ausgerichtet sind. Rotationen im Raum werden beschrieben als Rotationen um die jeweiligen Achsen. Welche Achse in welche Richtung zeigt ist nicht eindeutig definiert. Es gibt jedoch verschiedene Standards beziehungsweise Konventionen wie das links- oder rechtshändische Koordinatensystem. In ROS wird konventionell ein rechtshändisches Koordinatensystem, abgebildet in Abbildung 3.1 dargestellt. Dies wird im Folgenden ebenfalls als Standard verwendet.

Schreiben über Pfad (Indizierung, grobe Mathe), Posen, relative und absolute Transformationen

In der Robotik kommt es häufig vor, dass verschiedene (bewegliche) Komponenten relativ zu einem globalen Bezugssystem oder relativ zueinander beschrieben werden müssen. Die Bewegung eines übergeordneten Bezugssystems kann implizit für eine Veränderung der relativ zu diesem Bezugssystem platzierten Systeme führen. Dies lässt sich anhand eines Arm-Roboters zeigen, der mehrere miteinander verbundene Gelenke hat. Bewegt sich ein Gelenk werden automatisch auch



**Abbildung 3.1:** Schematische Darstellung der Konvention für Koordinatensysteme im ROS Framework. Die z-Achse zeigt nach oben, die x-Achse nach vorne und die y-Achse nach links. Die Rotation um die Achsen ist entsprechend der Konvention im Uhrzeigersinn.

die am Arm weiter außen befindlichen Gelenke mitbewegt. Aus Sicht des bewegten, übergordneten Bezugssystems hat sich die Position der untergeordneten Gelenke nicht verändert, aus Sicht des globalen Bezugssystems, wie zum Beispiel dem Montierungspunkt des Roboters, allerdings schon.

In ROS werden Anhängigkeiten zwischen Bezugssystemen in einer Baumstruktur, genannt *transformation tree (tf-tree)* dargestellt. Die Wurzel dieser Baumstruktur ist das globale Bezugssystem, wie zum Beispiel der Ursprung einer globalen Karte oder der Startpunkt der Trajektorie eines Roboters. Das globale Bezugssystem kann beliebig gewählt werden. Auf diese Weise kann ein Koordinatensystem, welches relativ zu einem anderen gelegen ist im Baum als Kindknoten seinem Bezugssystem untergeordnet werden. Es wird nur die relative Transformation (s. Kapitel 3.1.2) zwischen den Systemen im Baum gespeichert. Dies hat den Vorteil, dass bei der Bewegung eines Systems die untergeordneten Systeme nicht ebenfalls verändert werden müssen, da deren Transformationen relativ zum bewegten Bezugssystem angegeben sind und nicht global zur Wurzel des Baumes. Abbildung 3.2 zeigt ein Beispiel für einen Roboter mit mehreren voneinander abhängigen Systemen.

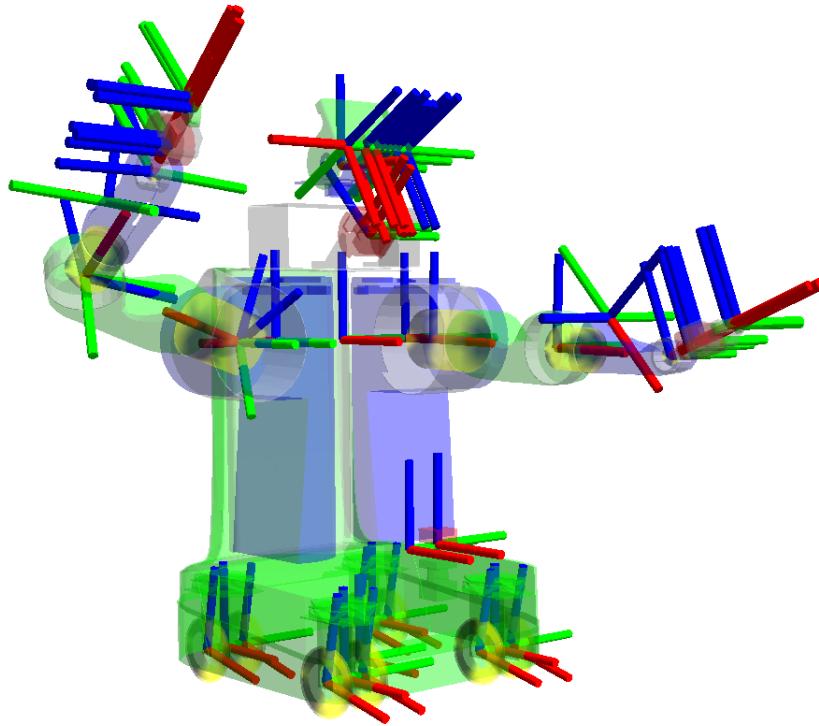
Auch räumliche Daten wie zum Beispiel Punktwolken aus Laserscannern können relativ zu verschiedenen Koordinatensystemen gesehen werden. So kann es nützlich sein die Punktwolke relativ zum Koordinatensystem des Scanners oder innerhalb des globalen Koordinatensystems zu betrachten. Um zwischen den Koordinatensystemen zu wechseln wird eine Koordinatensystemtransformation. Diese werden im folgenden Kapitel behandelt.

## Transformationen

Eine Koordinatensystemtransformation ist ein Sonderfall einer mathematischen Transformation, die eine Menge  $X$  auf sich selbst abbildet:

$$f : X \rightarrow X \quad (3.1)$$

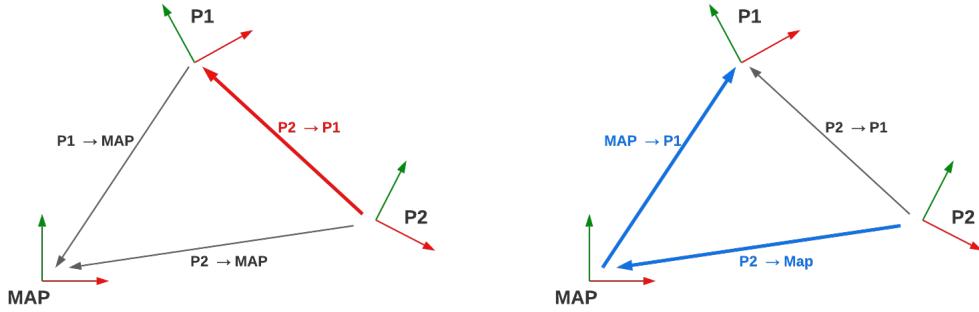
Eine Koordinatensystemtransformation beschreibt die Differenz zwischen zwei unterschiedlichen Koordinatensystemen und enthält sowohl die Translationsdifferenz als auch die Rotationsdifferenz. Zur Berechnung dieser Transformation zwischen zwei beliebigen Koordinatensystem  $C_1$  und



**Abbildung 3.2:** Schematische Darstellung eines Roboters und dessen beweglicher Teile. Die Ausrichtung der jeweiligen Gelenke wird mit einem lokalen Koordinatensystem beschrieben. Die globale Position einzelner Teile kann durch eine Verkettung der relativen Transformation in Richtung der Wurzel des Baumes bestimmt werden. Bild aus: [21]

$C_2$  wird die absolute Translation und Rotation beider Koordinatensysteme zum Ursprungskoordinatensystem, wie zum Beispiel den Ursprung einer Umgebungskarte, benötigt. Durch diese Rotations und Translationskomponenten beschreibt sich die absolute Position und Rotation der Koordinatensysteme im Raum aus Sicht des Ursprungskoordinatensystems  $C_{MAP}$ . Diese absolute Position und Rotation ist die Transformation von den jeweiligen Koordinatensystemen ins Ursprungskoordinatensystem.

Es existieren diverse Darstellungsweisen für Koordinatensystemtransformationen. Die intuitivste Weise der Darstellung ist die Darstellung als Vektor. In folgendem ist die Transformation vom Koordinatensystem  $C_X$  ins Koordinatensystem  $C_{MAP}$  dargestellt. Die Variablen  $t_i$  bezeichnen dabei die Translationskomponenten und die Variablen  $r_i$  die Rotationskomponenten um die jeweiligen Achsen.



**Abbildung 3.3:** Schematische Darstellung der bestimmung einer Transformation zwischen zwei Roboter-Posen  $P_1$  und  $P_2$ , hier zur Vereinfachung dargestellt in 2D. Gesucht ist die Transformation ( $T_{P_2 \rightarrow P_1}$ ), dargestellt im linken Teil der Abbildung als roter Pfeil. Diese kann implizit bestimmt werden durch eine Verkettung der Transformationen  $T_{P_2 \rightarrow MAP}$  und  $T_{MAP \rightarrow P_1}$ , hier dargestellt im rechten Teil der Abbildung in blau. Die Transformation  $T_{MAP \rightarrow P_1}$  ist dabei nicht explizit gegeben. Sie kann berechnet werden durch eine Inversion der Transformation  $T_{P_1 \rightarrow MAP}$ . Die finale Gleichung zur Berechnung der relativen Transformation ist dargestellt in Gleichung 3.3.

$$T_{C_X \rightarrow C_{MAP}} = \begin{pmatrix} t_x \\ t_y \\ t_z \\ r_x \\ r_y \\ r_z \end{pmatrix} \quad (3.2)$$

Neben der Vektordarstellung kann eine Transformation zusätzlich als eine  $4 \times 4$  Matrix dargestellt werden. Diese Darstellungsweise hat den Vorteil, dass die Transformation direkt per Matrixmultiplikation auf Daten wie um homogene Koordinaten erweiterte Punktdaten angewandt werden kann. Auch die direkte Kombination verschiedener Transformationen ist durch eine Matrixmultiplikation möglich. Hier gilt es zu beachten, dass Matrixmultiplikation nicht kommutativ ist und ein Vertauschen der Reihenfolge bei der Multiplikation zu unterschiedlichen Ergebnissen führen kann. Bei einer Verkettung von Transformationen durch Multiplikation wird die Matrix zuerst angewandt, welche am Ende der Multiplikation steht.

In dieser Arbeit werden Transformationen zum Beispiel verwendet um zu bestimmen, wie sich ein Roboter zwischen zwei Messungen bewegt hat. Diese Transformationen beschreiben relative Differenzen zwischen Roboterpositionen im Raum. Diese Roboterpositionen werden auch **Posen** genannt. Eine Pose ist dabei eine meist absolute Beschreibung der Translation und Rotation eines Roboters zu einem gewissen Zeitpunkt  $t$  aus Sicht des Ursprungskoordinatensystems wie zum Beispiel dem Map-Ursprung  $C_{MAP}$ .

Abbildung 3.3 zeigt die Mathematik hinter der Berechnung der Posedifferenz exemplarisch. Es wird deutlich, dass eine gesuchte Transformation aus dem Koordinatensystem von Pose  $P_2$  in das Koordinatensystem von Pose  $P_1$  ( $T_{P_2 \rightarrow P_1}$ ) gegeben ist durch:

$$(T_{P_2 \rightarrow P_1}) = (T_{P_1 \rightarrow MAP})^{-1} * T_{P_2 \rightarrow MAP} \quad (3.3)$$

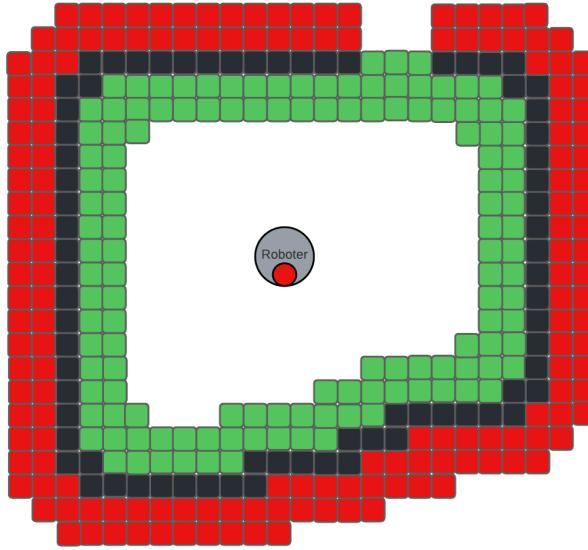
Basierend auf den erläuterten mathematischen Grundlagen wird in Kapitel 3.3 die Grundlagen von SLAM, insbesondere von TSDF basierten SLAM Verfahren, erörtert. Zuvor werden in nachfolgender Sektion die Eigenschaften und Anwendungsbereiche der TSDF beschrieben.

## 3.2 TSDF

Zur Lösung des **Simultaneous Localization and Mapping (SLAM)** (siehe Kapitel 3.3) Problems in unbekannten Umgebungen wird im Regelfall eine Form der Kartenrepräsentation und die Algorithmik benötigt sich auf Basis der Kartenrepräsentation zu lokalisieren und diese im Anschluss aktualisieren zu können. Bei vielen Lösungsansätzen wie zum Beispiel einer inkrementellen **Registrierung** (siehe Kapitel 3.3) mit dem **Iterative Closest Point (ICP)** [2] oder **Generalized Iterative Closest Point** [22] Algorithmus werden als Kartenrepräsentation registrierte Punktwolken verwendet. Punktwolken stellen dabei keine geschlossenen Oberflächenrepräsentationen dar und benötigen viel Speicher im Gegensatz zu einigen geschlossenen Repräsentationen wie aus den Punktwolken generierten Dreiecksnetzen. Ein großer Nutzen einer solchen Repräsentation ist die vereinfachte Lokalisierung und Navigation auf Basis der Kartenrepräsentation.

Eine weitere Form der geschlossenen Oberflächenrepräsentation der Umgebung sind **Signed Distance Fields (SDF)**. Im Gegensatz zu Dreiecksnetzen sind die SDF implizite, volumetrische Beschreibung der Oberfläche [25]. Sie beschreiben die Oberfläche nicht direkt, sondern den Raum um die Oberfläche herum. Die **Signed Distance** ist die orthogonale metrische Distanz eines beliebigen Punktes  $p$  zur Oberfläche räumlicher Daten, wie zum Beispiel Punktwolken. Diese Distanz kann sowohl negativ, als auch positiv sein. Unterschieden wird zwischen dem Innenbereich, räumlich gesehen vor einer Wand oder einem Hindernis, und dem Außenbereich, welcher räumlich gesehen hinter dem vom Laser getroffenen Objekt befindlich ist. Abbildung 3.4 zeigt ein zweidimensionales, schematisches Beispiel für eine diskretisierte TSDF nach Abtasten der Oberfläche einer unbekannten Umgebung durch einen Laserscanner.

Die **Truncated Signed Distance Function (TSDF)** ist eine Unterklasse der SDF. Sie betrachtet die Distanz zur Oberfläche nur bis zu einer maximalen Distanz  $tsdf_{max}$ , auch  $\tau(\tau)$  genannt[10]. Alle Werte die weiter von der Oberfläche entfernt oder unbekannt sind, erhalten als Wert  $\tau$  selbst. Dies spart Rechenaufwand, da nur die Werte in direkter Nähe zur Oberfläche angepasst werden müssen. In Gebieten, in denen der TSDF-Wert  $\tau$  entspricht, kann jedoch keine Aussage getroffen werden, wo die nächste Oberfläche ist, oder wie weit sie entfernt ist. Es ist lediglich bekannt, dass die betrachtete Position nicht in direkter Nähe zur Oberfläche befindlich und mindestens  $\tau$  entfernt ist. Das Intervall möglicher Werte der TSDF ist:



**Abbildung 3.4:** Schematische Darstellung einer diskretisierten 2D TSDF Karte. Abgebildet sind nur Voxel mit TSDF-Werten im Intervall  $]-\tau, \tau[$ . In der Mitte der Karte befindet sich ein Roboter mit einem Laserscanner (hier dargestellt in rot). Der Roboter befindet sich zum Beispiel in einem Raum. Der Innenbereich (des Raumes) mit positiven TSDF-Werten ist hier dargestellt in grün, der Außenbereich mit negativen TSDF-Werten in rot. Die Oberfläche, in deren Umgebung die TSDF-Werte nahezu Null sind, ist abgebildet in schwarz.

$$I = [-\tau, \tau] := \{x \in \mathbb{R} | \tau > 0\} \quad (3.4)$$

Über SDF und TSDF können kontinuierliche Karten erstellt werden. Da kontinuierliche Karten aber unendlich viel Speicherplatz benötigen, wird der Raum diskretisiert. Die Diskretisierung erfolgt durch eine Aufteilung der Umgebung in **Voxel** mit definierbarer, fester Seitenlänge  $v_{res}$  [26] [10]. Jeder Voxel enthält einen approximierten (T)SDF-Wert. Diese Form der Darstellung kann auch als pseudo-kontinuierlich angesehen werden, da durch eine Approximation über benachbarte Zellen ein approximierter TSDF-Wert für jeden beliebigen Raumpunkt berechnet werden kann. Dadurch sind TSDF basierte Karten ideal, um mittels des **Marching Cubes Algorithmus** [15] eine polygonale Netz-Repräsentation, wie zum Beispiel ein Dreiecksnetz generieren zu können. Dieses kann zum Beispiel als optische Referenz für die Qualität der Karte verwendet werden.

Diese Arbeit beschäftigt sich mit einer möglichen Integration von Schleifenschlüssen in einen auf einer TSDF-Karte basierenden Ansatz wie vorgestellt von Eisoldt et al. [10]. Der Aufbau und die Verwendung der TSDF-Karte ist im nachfolgenden Abschnitt beschrieben. Ziel ist die Korrektur von Fehlern bei der Registrierung und der damit verbundenen Korrektur der TSDF-Karte. Dies ist in Kapitel 5 und 6 beschrieben.

### 3.2.1 TSDF-Karte

Eisoldt et al. [10] basieren ihren SLAM Ansatz auf einer diskreten, inkrementell erweiterten TSDF Karte. Zur Registrierung (vergleiche Kapitel 3.3) verwenden sie ebenfalls die TSDF-Karte. Punktwolken werden dabei mit einer **Point-to-TSDF** Strategie an die TSDF Karte registriert [10]. In [10] werden neue Punktdaten nicht an die globale TSDF Karte registriert sondern an eine lokale TSDF Karte fester Größe. Lediglich die lokale Karte befindet sich im Arbeitsspeicher und lädt wenn nötig Daten aus der globalen Karte, die durch einer **HDF5-Datei** repräsentiert ist und auf der Festplatte gespeichert ist, nach. Das **Hierarchical Data Format 5 (HDF5)** [11] ist ein Dateiformat für die Speicherung und Verwaltung von Daten in einem hierarischen System, das dem Dateisystem von Windows oder UNIX Betriebssystemen ähnelt. HDF5 erlaubt die Gruppierung von zusammengehörigen Daten und die Speicherung von Metadaten wie zum Beispiel den Hyperparametern der verwendeten Karte. Auch Schachtelungen sind möglich. HDF5 eignet sich besonders für die effiziente Serialisierung und Deserialisierung komplexer Daten oder Objekte und kann die interne Struktur der zu serialisierenden Objekte abbilden. Diese Aufteilung in eine globale und eine lokale Karte sorgt dafür, dass [10] auch für große Umgebungen (**Large-Scale**) geeignet ist und der Arbeitsspeicher nicht überläuft. Die Implementation dieser TSDF-Kartenstruktur wird auch in dieser Arbeit verwendet. Sie dient als Basis für die Bestimmung der Datenassoziationen wie beschrieben in Kapitel ?? und bildet die Grundlage für jegliche Form der Kartenoptimierung, die in dieser Arbeit vorgenommen wird.

Die globale TSDF-Karte ist unterteilt in Teilstücke beziehungsweise **Chunks** fester Größe, deren Seitenlänge durch die Anzahl der TSDF-Voxel entlang einer Seitenlänge definiert ist. Bei Bedarf kann ein beliebiger Chunk in den Arbeitsspeicher geladen oder zurück auf die Festplatte in die HDF5-Datei geschrieben werden. Diese enthält in der hierarchischen Struktur ein Datenset für jeden angelegten Chunk. Neben den TSDF-Informationen ist nach Durchlaufen des SLAM-Algorithmus zusätzlich die erstellte Pose-Historie in der HDF5 gespeichert. Abbildung 3.5 zeigt den internen Aufbau der HDF5 Datei, in der die globale Karte abgespeichert ist.

Die Bezeichnung (TSDF-)Karte beziehungsweise **Map** wird im Folgenden für die implizite TSDF Repräsentation der Umgebung verwendet. Der Ursprung der Karte ist im Folgenden als Ursprung des Weltkoordinatensystems definiert.

### 3.2.2 TSDF-Update

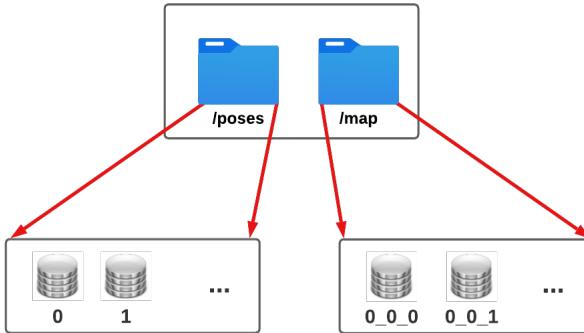
TSDF-Update

erklären.

Nachfolgende Sektion behandelt das Thema SLAM und gibt einen groben Überblick über einige SLAM Ansätze.

## 3.3 SLAM

Diese Sektion befasst sich mit den Grundlagen von SLAM, stellt heraus welche Varianten von SLAM Verfahren es gibt und wie sich TSDF basierte Verfahren, insbesondere der HATSDF-SLAM Ansatz von Eisoldt et al. ?? von diesen unterscheidet.



**Abbildung 3.5:** Innere Struktur der HDF5-Datei, die die globale TSDF-Karte repräsentiert. Auf der obersten hierarchischen Ebene werden die Daten in zwei Gruppen aufgeteilt. */poses* enthält dabei die Trajektorie des Roboters, separiert in einzelne Posen, die mit Null beginnend indiziert sind. */map* enthält die eigentliche TSDF-Karte. Sie ist in unterteilt in Chunks, die ein Datenarray aus Paaren von TSDF-Wert und TSDF-Gewicht enthalten. Das Label der Chunks entspricht den diskreten Koordinaten der Chunks im Raum.

SLAM ist der Prozess der simultanen Generierung einer Karte einer unbekannten Umgebung und der Lokalisierung innerhalb dieser Karte beziehungsweise Umgebung. SLAM ist ein *Henne-Ei-Problem*, da auf der einen Seite eine vollständige Karte benötigt um die Pose des Roboters akkurat zu bestimmen, auf der anderen Seite allerdings eine akkurate Posenhistorie benötigt wird um eine gute Karte der Umgebung aufbauen zu können. Eine grobe Übersicht über existierende SLAM Verfahren liefert der Stand der Forschung in Kapitel ???. Im Folgenden werden einige der genannten Verfahren erneut aufgegriffen und erläutert. Schlussendlich wird erklärt, welche der SLAM Verfahren für diese Masterarbeit von Interesse sind und wie sie genutzt werden.

Der **Iterative Closest Points (ICP)** Algorithmus nach Besl und McKay [2] ist ein Algorithmus zur **Registrierung** von Punktwolken. Als **Registrierung** wird der Prozess der Zusammenführung von Punktwolken bezeichnet, die von unterschiedlichen Orten aus aufgenommen werden. Um Punktwolken möglichst gut zusammenzuführen, wird versucht die aus den Laserscans entstandenen Punktwolken maximal zu überlappen. Dieser Prozess wird auch **Scan Matching** genannt. Beim Scan Matching wird zwischen dem **Model** und dem **Scan** unterschieden. Als Scan bezeichnet werden die Daten, die an das Model registriert werden sollen. Das Ergebnis des Scan Matching ist eine Approximation der Transformation  $T_{Scan \rightarrow Model}$  zwischen den Posen von denen aus die Punktwolken aufgenommen wurden. Damit untersucht werden kann, wie gut diese Approximation ist, liefern ICP und verwandte Algorithmen wie zum Beispiel **Generalized Iterative Closest Points** [22] ein Maß für die Genauigkeit der Approximation. Dies ist im Fall von ICP und GICP der sogenannte **Fitness-Score**. Er beschreibt die durchschnittliche quadrierte Distanz zwischen den **Nearest Neighbors (nächsten Nachbarn)** der Punktwolken nach Anwendung der approximierten Transformation  $T_{Scan \rightarrow Model}$  der Scanpunktfolge in das Koordinatensystem der Modellpunktfolge. Er gibt dementsprechend an, wie groß die durchschnittliche quadrierte Distanz eines Punktes aus der Modellpunktfolge zum euklidisch nächsten Punkt der transformierten Scanpunktfolge ist.

Sowohl ICP, als auch GICP sind inkrementelle Algorithmen, dass heißt sie nähern sich inkrementell einem Optimum immer weiter an. Dabei wird die approximiert Transformation jeweils um ein  $\delta T$  verändert. Fällt dieses  $\delta T$  in einer Iteration unter einen vom Benutzer gewählten Schwellwert, oder wird eine maximale Anzahl an Iterationen erreicht, bricht der Algorithmus ab und gibt die finale Transformation zurück. Genanntes Optimum ist dabei im Regelfall allerdings kein globales, sondern lediglich ein lokales Optimum aus dem weder ICP noch GICP herauskommen, sobald sie hineingeraten. Aus diesem Grund gilt es die jeweiligen Ausgaben der Algorithmen zum Beispiel basierend auf dem resultierenden Fitness-Score zu analysieren.

Beide Algorithmen werden in Kapitel 3.4 und Kapitel 5 in Bezug auf die Identifikation von Loop-Closures evaluiert.

Varianten (auf die eingegangen wird):

ICP Graph-SLAM - $\zeta$  Global Relaxation

Einzelne Verfahren näher beschreiben, erklären welche Bibliotheken verwendet werden

1. Grundlagen von SLAM beschreiben - $\zeta$  Varianten des SLAM - $\zeta$  Bezug zu HATSDF-SLAM
2. Voraussetzungen (Repräsentationen für Posen (Pfad) und Umgebung)
3. Überleitungen in weitere Sektionen machen - $\zeta$  Loop Closure als mögliche Verbesserung des SLAM - $\zeta$  TSDF als Kartenrepräsentation - $\zeta$  auf Vorteile von TSDF eingehen (z.B. einfache Integration in Marching Cubes Algorithmus)

## 3.4 Loop Closure

Warum wird Loop Closure benötigt? Welchen Mehrwert gibt es? Wie wäre ein grundlegendes vorgehen? verweisen auf Loop-Closure Kapitel



# Kapitel 4

## Datenassoziationen

Wie in der Einleitung beschrieben, soll in einem ersten Ansatz analysiert werden, ob eine TSDF Karte mit gegebenem initialen Pfad durch die Optimierung des initialen Pfades mittels Schleifenschlüssen verbessert werden kann. Dazu ist im ersten Schritt zu identifizieren, welcher Teil der Karte mit welcher Pose assoziiert ist um bei einer Veränderung der Trajektorie entscheiden zu können, wie die Karte angepasst werden muss. Dies Kapitel befasst sich mit der Generation von Datenassoziationen zwischen den Posen des Pfades und der TSDF-Karte.

### 4.1 Ansatz

Wie bereits in Kapitel 3.2 beschrieben, wird die TSDF Karte in [10] inkrementell erweitert, sobald eine definierte minimale Distanz zurückgelegt wurde. Dabei werden nicht nur neue Zellen beschrieben, sondern auch die Werte bereits beschriebener Zellen gewichtet verändert. Dementsprechend kann die Information in einer Zelle eine Akkumulation beliebig vieler Updates sein und beliebig vielen Posen zugeordnet werden. Diese Information gilt es zu berücksichtigen, wenn auf Basis einer gegebenen TSDF Karte Datenassoziationen identifiziert werden sollen. Eine Möglichkeit der Generation dieser Assoziation wäre eine **1:1** Beziehung zwischen den Zellen und Posen aufzubauen. Dann würde eine Zelle maximal einer Pose zugeordnet werden. Da bereits bekannt ist, dass eine Zelle von mehreren Posen angepasst werden kann, ist diese Art der Beziehung zwischen Posen und TSDF-Zellen allerdings von einem großen Informationsverlust geprägt. Die Alternative zur **1:1** Beziehung ist eine **1:N** Beziehung zwischen einer Zelle und  $N$  Posen. Diese Beziehung ist aufgrund der genannten Eigenschaften des TSDF-Karten Updates der **1:1** Beziehung zu bevorzugen.

Die Informationen darüber, welche Position welche TSDF Zelle beschrieben lässt sich allerdings nicht ohne Weiteres aus der TSDF Karte herauslesen. Um dies zu ermöglichen könnte [10] um die Funktion erweitert werden an jeder Zelle zusätzlich ein Array zu speichern, in dem die Posen enthalten sind, die die betroffene Zelle modifiziert haben. Dieses Array muss in einem eigenen Datenset gespeichert sein, da die Anzahl Posen, die auf diese Weise einer Zelle zugeordnet werden können, dynamisch ist. Das bedeutet, dass für jede einzelne TSDF Zelle

ein eigenes Datenset gespeichert werden muss, in dem die zugehörigen Posen enthalten sind. Je nach Auflösung der diskretisierten Karte müssten nach diesem Ansatz mehrere Millionen separate Datensets gespeichert werden. Ein solches Vorgehen erfordert nicht nur viel Speicher, sondern ist auch aus hierarchischer Betrachtungsweise keine sinnvolle Herangehensweise. Eine Möglichkeit, die gewünschten Daten auf Basis einer gegebenen TSDF Karte zu generieren ist diese über die Methode, mit der die Daten generiert wurden, zu regenerieren. In [10] wird die TSDF-Karte über ein **Ray-Marching** generiert. Eine alternative zum Ray-Marching stellt der Bresenham Algorithmus dar, der die Diskretisierung der Karte ausnutzt. Beide Varianten werden im Folgenden beschrieben, evaluiert und miteinander verglichen. Zunächst beschreibt der folgende Abschnitt die Serialisierung der Datenassoziation in der HDF5-Datenstruktur.

## 4.2 Serialisierung

Diese Sektion beschreibt, wie identifizierte Assoziationen in der hierarchischen Struktur der HDF5-Datei gespeichert werden, die Daten der TSDF-Karte, sowie die zugehörige Pose-Historie enthält. Wie in Kapitel 3.2 beschrieben, enthält die HDF5 Struktur mehrere Gruppen, die jeweils weitere Daten enthalten. Zu diesen Gruppen gehören in diesem Fall */map* und */poses*. Die Map-Gruppe enthält die serialisierten TSDF-Zellenwerte und TSDF-Zellengewichte. Die Poses-Gruppe enthält die serialisierten 6D Posen des Pfades als Datensets. Der HDF5 interne Pfad eines dieser Pose-Datensets ist */poses/[index]*, wobei *index*, der Index der Pose im Pfad ist. An dieser Stelle wird nun eine Erweiterung vorgenommen um die generierten Assoziationen zu serialisieren. Anstelle der Datensets wird für jede Pose eine eigene Gruppe erstellt. Diese Gruppe erhält als Namen ebenfalls den Pfadindex der Pose. Innerhalb dieser Gruppe wird ein Datenset für die Pose und optional ein weiteres Datenset für die Datenassoziationen angelegt, sofern erforderlich. Abbildung 4.1 zeigt die neue interne HDF5 Struktur nach dieser Änderung.

Auf Basis dieser Änderungen wird im Folgenden erläutert, wie die zu speichernden Zellen für jede Pose ermittelt werden.

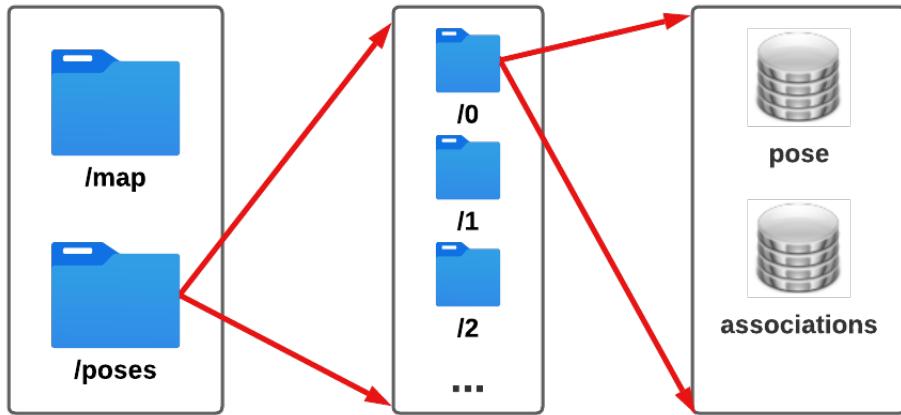
## 4.3 Algorithmen

Diese Sektion stellt die Algorithmen heraus, mit denen beschriebene Assoziationen identifiziert werden können. Die Ergebnisse der Algorithmen werden miteinander verglichen und evaluiert.

### 4.3.1 Ray-Tracing

Eine Möglichkeit der Ermittlung der mit einzelnen Posen assoziierten Teilbereiche der TSDF Karte ist die die Erstellung eines künstlichen Laserscans innerhalb der TSDF Karte, ausgehend von der entsprechenden Pose. Entsprechend wurde im Zuge dieser Arbeit ein **Ray-Tracer** entwickelt, der künstliche Laserstrahlen innerhalb der TSDF-Karte aussendet und die Schnittpunkte mit der TSDF Karte überprüft. Der Ray-Tracer ist beliebig konfigurierbar ist und kann an die Parameter verschiedenster Laserscanner angepasst werden. Die wesentlichen Parameter und deren Bedeutung sind Tabelle 4.1 zu entnehmen.

TODO: anhand mehrerer Datensätze Assoziationen bilden und schauen, von wie vielen Posen eine Zelle im Durchschnitt angepasst wurde und hochrechnen, was das für den Speicher bedeutet



**Abbildung 4.1:** Schematische Darstellung der HDF5 internen Datenstruktur nach Speicherung der generierten Datenassoziationen zwischen TSDF-Zellen und Posen. Die in Kapitel 4.1 beschrieben  $1:N$  Beziehung zwischen einer Zelle und den zugehörigen Posen ist hier indirekt realisiert. Anstelle pro Zelle ein Datenset zu erstellen, wir für jede Pose ein Datenset enthält, das alle assoziierten TSDF-Zellen enthält. Verschiedene Posen können dabei dieselbe TSDF-Zelle assoziieren. Das Datenset `pose` enthält die Transformation der aktuellen Pose ins Ursprungskoordinatensystem. Das Datenset `associations` enthält ein Array der assoziierten Zellkoordinaten, die durch die Diskretisierung ganzzahlig sind und als Integer abgespeichert werden.

Zur Emulation des Laserscans wird zunächst ein Array erstellt, in dem die aktuellen Endpunkte der jeweiligen Rays gespeichert werden. Die Anzahl an Endpunkten  $n$  ist definiert durch die konfigurierte Auflösung. Sie beträgt:

$$n = \text{vert\_res} \cdot \text{hor\_res} \quad (4.1)$$

Der Startpunkt jedes Rays ist die Pose  $P_i$ , von der aus der Laserscan ausgesendet wird. Ziel ist in jeder Iteration alle Rays um `step_size` zu verlängern und die TSDF-Zellen zu evaluieren, die derzeit von den einzelnen Rays getroffen werden. Für diese Verlängerung der Rays müssen diese zunächst initialisiert werden. Diese Initialisierung erfolgt auf Basis der parametrisierten Öffnungswinkel des Laserscanners `opening_degree_vert` und `opening_degree_hor`, sowie der konfigurierten vertikalen und horizontalen Auflösung `vert_res` und `hor_res`. Zunächst werden die Winkelbereiche definiert, in den der Ray-Tracer operiert. Diese setzen sich aus den Öffnungswinkeln zusammen. Der Winkelbereich in horizontaler Richtung beträgt:

$$I_{\text{hor}} = [-\text{opening\_degree\_hor}, \text{opening\_degree\_hor}] \quad (4.2)$$

Der Winkelbereich in vertikaler Richtung beträgt:

$$I_{\text{vert}} = [-\text{opening\_degree\_vert}, \text{opening\_degree\_vert}] \quad (4.3)$$

**Tabelle 4.1:** Parameter des in dieser Arbeit entwickelten Ray-Tracers zur Bestimmung des mit einer beliebigen Pose assoziierten Teilbereichs der TSDF-Karte. Der horizontale Öffnungswinkel wird an dieser Stelle als 360 Grad angenommen.

Parameter	Funktionsweise	Default-Wert
<i>opening_degree</i>	Definiert den vertikalen Öffnungswinkel des Ray-Tracers. Anzugeben in Grad.	45
<i>hor_res</i>	Definiert die horizontale Auflösung des Laserscanners. Der gegebene Wert entspricht der Anzahl <i>Rays</i> pro Scanebene.	1024
<i>vert_res</i>	Definiert die vertikale Auflösung des Laserscanners. Der gegebene Wert entspricht der Anzahl an Scanebenen im Laserscan.	128
<i>step_size</i>	Definiert, wie groß die Schrittweite beim Aussen-den der einzelnen Rays ist. Der Wert ist in Metern anzugeben. Der Default-Wert ist direkt an die Zellgröße der diskreten TSDF-Karte <i>map<sub>res</sub></i> gekoppelt und beträgt $\frac{map_{res}}{2}$ .	0.032
<i>ray_size</i>	Definiert die Dicke des Strahls in der Visualisierung. Dieser Parameter dient lediglich zur erleichterten Visualisierung des Laserscans bei unterschiedlicher Konfiguration. Der Wert ist in Metern anzugeben.	0.01

Die jeweiligen Winkelbereiche werden durch die konfigurierte Auflösung diskretisiert. Die horizontale Schrittweite des Laserscanners beträgt:

$$\Delta_{hor} = \frac{opening\_degree\_hor}{hor\_res} \quad (4.4)$$

Die vertikale Schrittweite des Laserscanners beträgt:

$$\Delta_{vert} = \frac{opening\_degree\_vert}{vert\_res} \quad (4.5)$$

Basierend auf den unteren und oberen Winkelschranken und der berechneten Schrittweite zwischen diesen Schranken kann nun das Array initialisiert werden. Dazu wird in zwei Schleifen über die beiden Winkelintervalle  $I_{vert}$  und  $I_{hor}$  iteriert und der aktuelle Wert jeweils um die berechneten Delta  $\Delta_{vert}$  und  $\Delta_{hor}$  inkrementiert. Aus den beiden Winkeln  $\alpha$  und  $\beta$  der aktuellen Iteration der Schleifen, sowie einer beliebigen Distanz initialen Länge des Rays, wie zum Beispiel der Schrittweite *step\_size* können nun für jeden Punkt die initialen Ray-Punkte berechnet werden, die den Richtungsvektor des Rays definieren. Hierzu ist eine Umwandlung von Kugelkoordinaten in das Kartesische Koordinatensystem notwendig. Mit *alpha*, *beta* und

*step\_size* wird in Kugelkoordinaten genau ein Punkt im dreidimensionalen Raum beschrieben. Um diese in kartesische Koordinaten im ROS Koordinatensystem umzuwandeln wird folgende Formel verwendet ( $\alpha$  und  $\beta$  gegeben in Radianen,  $\alpha$  beschreibt den aktuellen Winkel um die z-Achse,  $\beta$  die aktuelle Rotation um die y-Achse):

$$\begin{pmatrix} x_{P_i} \\ y_{P_i} \\ z_{P_i} \end{pmatrix} = step\_size \cdot \begin{pmatrix} \cos(\alpha) \cdot \cos(\beta) \\ \sin(\alpha) \cdot \cos(\beta) \\ \sin(\beta) \end{pmatrix} \quad (4.6)$$

Der Punkt  $\begin{pmatrix} x_{P_i} \\ y_{P_i} \\ z_{P_i} \end{pmatrix}$  beschreibt hier zunächst nur den Ray-Punkt aus Sicht des lokalen Map-Koordinatensystems, das durch  $P_i$  beschrieben ist. Um diesen aus Sicht des globalen Koordinatensystems  $\mathbb{M}$  zu betrachten, muss dieser Punkt dorthin transformiert werden. Grundlagen zur Transformation werden in Kapitel 3.1.2 behandelt. Es ist essentiell, dass an dieser Stelle nicht nur die Translation, sondern auch die Rotation berücksichtigt wird um den Scan

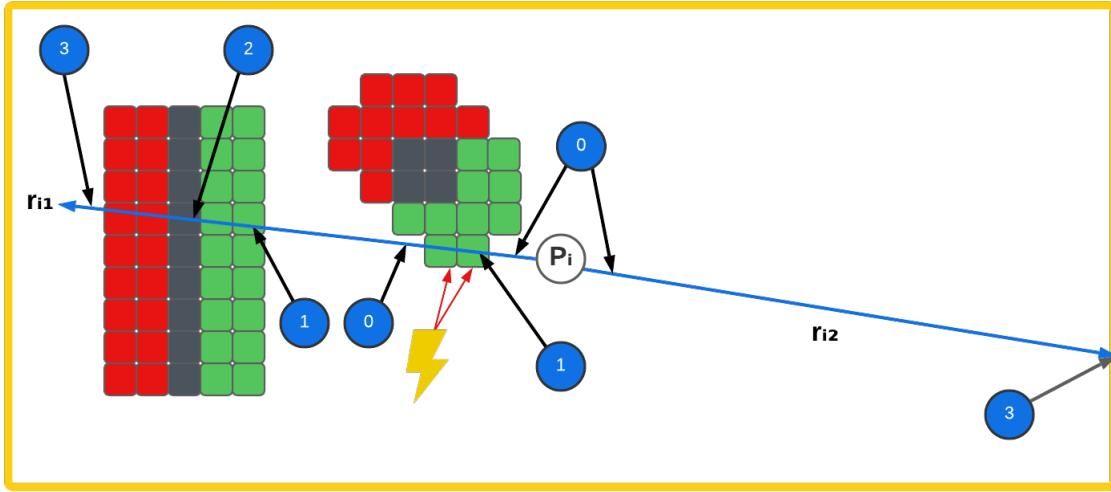
von Pose  $P_i$  bestmöglich replizieren zu können. Die Transformation des Vektors  $\begin{pmatrix} x_{P_i} \\ y_{P_i} \\ z_{P_i} \end{pmatrix}$  vom Koordinatensystem beschrieben durch Pose  $P_I$  in das globale Koordinatensystem  $\mathbb{M}$  mit der Transformationsmatrix  $T_{P_i \rightarrow \mathbb{M}}$  ist gegeben durch:

$$\begin{pmatrix} x_{\mathbb{M}} \\ y_{\mathbb{M}} \\ z_{\mathbb{M}} \end{pmatrix} = T_{P_i \rightarrow \mathbb{M}} \cdot \begin{pmatrix} x_{P_i} \\ y_{P_i} \\ z_{P_i} \end{pmatrix} \quad (4.7)$$

Auf diese Weise werden alle initialen Endpunkte des emulierten Laserscans berechnet. Auf Basis der berechneten initialen Endpunkte und des bekannten Anfangspunktes gegeben durch den Translationsanteil von  $P_i$  kann das inkrementelle Ray-Tracing beginnen. In jeder Iteration des Ray-Tracing werden alle Rays um *step\_size* verlängert und die entsprechend getroffenen Zellen evaluiert. Um einen Vektor  $\vec{v}$  gegeben durch den Translationsanteil  $\vec{t}_i$  und den aktuellen Endpunkt des betrachteten Rays  $\vec{r}_i$  um *step\_size* zu verlängern und daraus den neuen Endpunkt des Rays  $\hat{\vec{r}}_i$  zu berechnen wird folgende Formel verwendet:

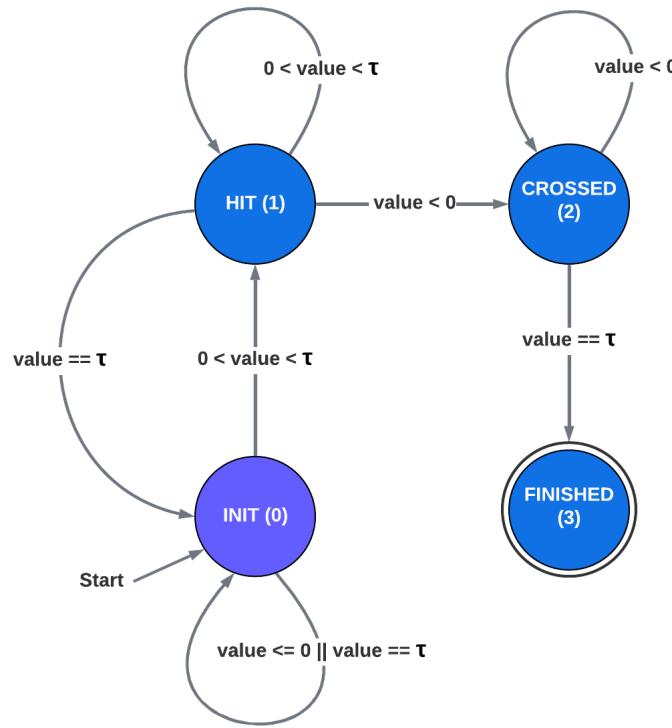
$$\hat{\vec{r}}_i = \frac{\|\vec{r}_i - \vec{t}_i\| + step\_size}{\|\vec{r}_i - \vec{t}_i\|} \cdot (\vec{r}_i - \vec{t}_i) + \vec{t}_i \quad (4.8)$$

Nach der Verlängerung eines Rays  $\vec{r}_i$  wird die in der aktuellen Iteration  $j$  getroffene TSDF-Zelle  $C_i^j$  evaluiert. Je nach Schrittweite *step\_size* und Auflösung des Ray-Tracers ist es möglich,



**Abbildung 4.2:** Schematische Darstellung 2D Darstellung der verschiedenen Zustände eines einzelnen Rays des Ray-Tracers innerhalb einer TSDF-Darstellung. Negative TSDF-Werte dargestellt in rot, positive in grün. Der approximierte Nulldurchgang in der TSDF ist hier gräulich dargestellt, die umgebende lokale Karte in gelb. Ausgehend von Pose  $P_i$  sind zwei Rays  $r_{i1}$  und  $r_{i2}$  dargestellt, die die verschiedenen Fälle abdecken, die es zu berücksichtigen gilt. Eine genaue Beschreibung der Zustandsänderungen der Rays im Zustandsdiagramm 4.3 zu entnehmen. Die entsprechenden Zustände sind dargestellt als blaue Kreise, die die jeweilige Zustandsnummer enthalten. Die entsprechenden Definitionen der Zustände sind ebenfalls im Zustandsdiagramm 4.3 zu entnehmen. Die mit einem Blitz markierten Zellen werden zwar von dem ausgesandten Ray  $r_{i1}$  getroffen, dürfen allerdings aufgrund der Evidenz im aktuellen Ray nicht mit der Pose assoziiert werden, da im Anschluss an diese Zellen kein Nulldurchgang, sondern Freiraum folgt. Der Freiraum ist hier in weiß dargestellt und setzt sich aus den TSDF-Zellen zusammen, die Default-Werte enthalten und entsprechend außer der minimalen Entfernung  $\tau(\tau)$  zur Oberfläche, keine räumlichen Informationen besitzen. Gleicher Ausnahmefall tritt ein, wenn der Ray lediglich negative TSDF-Zellen trifft. Diese werden ebenfalls nicht aufgrund der Evidenz des betrachteten Rays mit der Pose assoziiert.

dass  $C_i^j$  bereits evaluiert wurde und schon eine Assoziation mit der Pose  $P_i$  hergestellt ist. Um diesen Fall zu überprüfen und zu verhindern, dass duplizierte Assoziationen gespeichert werden, wird eine Hash-Map genutzt, deren Hash auf Basis der Koordinaten der TSDF-Zelle berechnet wird. Ist  $C_i^j$  bereits in der Hash-Map gespeichert, ist der aktuell betrachtete Ray  $\vec{r}_i$  für diese Iteration fertig evaluiert und der nächste Ray kann betrachtet werden. Um zu entscheiden ob eine nicht assoziierte Zelle  $C_i^j$  als Assoziation in Frage kommt müssen mehrere Zustände des Rays definiert werden. Abbildung 4.2 zeigt die benötigten Zustände und die Bedingungen für einen Wechsel des Status gegeben den aktuellen Status und die betrachtete Zelle  $C_i^j$ , sowie dieren TSDF-Wert und TSDF-Gewicht. Ein Ray ist beschränkt durch die lokale Karte um  $P_i$  3.2.1, sowie die Struktur der TSDF-Karte. Detektiert ein Ray einen Wechsel von positive auf negative TSDF-Werte (**Nulldurchgang**) in der TSDF, stoppt der Ray-Tracer, sobald er erneut positive Werte detektiert. Diese Herangehensweise sorgt dafür, dass mit der Pose  $P_i$  keine Zellen assoziiert werden, die von dieser Pose aufgrund der Begrenzungen der lokalen Karte nicht gesehen werden konnten oder hinter Wänden befindlich sind.



**Abbildung 4.3:** Zustandsdiagramm der internen Zustände eines Rays. Zustandsübergänge sind beschrieben durch einen initialen Zustand und den TSDF-Wert der aktuellen Zelle ( $value$ ). In Zustand 1 werden gefundene Assoziationen zunächst nicht abgespeichert, da noch nicht bekannt ist, ob diese Zellen zu einem Nulldurchgang gehören oder ob der Ray nur Zellen kreuzt, die von einer anderen Pose aus befüllt wurden. Je nach TSDF-Wert der aktuellen Zelle werden die zwischengespeicherten Assoziationen aus Zustand 1 entweder verworfen oder in der HDF5 gespeichert.

Die Ergebnisse dieses Ansatz sind in Abbildung 4.5 im Vergleich mit den Ergebnissen des Bresenham Algorithmus dargestellt, der in der nachfolgenden Sektion behandelt wird. In dem genutzten Datensatz können nur etwa 91% der Zellen assoziiert werden. Diese Zahl ähnelt auch der von Bresenham. . Sektion 4.3.4 evaluiert die Ergebnisse von Ray-Tracing und Bresenham und vergleicht diese miteinander. Zudem wird Bezug zum Informationsverlust bei der Assoziationsidentifikation genommen.

Für mehrere Datensätze zahlen bilden, graphisch darstellen

### 4.3.2 Bresenham

Eine alternative algorithmische Herangehensweise an das beschriebene Problem der Assoziationsidentifikation ist die Nutzung des Bresenham-Algorithmus nach Bresenham [5]. Dieser wurde ursprünglich verwendet, um einen digitalen Plotter mittels eines Computers zu kontrollieren und beliebige zweidimensionale Linien und Kurven approximativ abzubilden. Der Plotter lässt sich dabei in acht Richtungen auf einem diskretisierten Raster bewegen. Bresenham [5] beschreibt, wie sich die Zellen im Raster berechnen lässt, die das gegebene Liniensegment einer Kurve oder eine

Beschreibung von Türen, Beschreibung von Problem wie Verdeckung durch Diskretisierung, ggf. Ray-Trace Bild, Beschreibung von Fallstrichen: Nicht getroffene Zellen, Information loss

ggf. RVIZ-Bild des Ray-Tracing Markers

Tür-Problem



**Abbildung 4.4:** Schematische Darstellung des Bresenham-Algorithmus in zwei Dimensionen. Der Bresenham Algorithmus bestimmt, welche Voxel die Linie zwischen einem gegebenen Startvoxel und einem Endvoxel am besten beschreiben.

Linie am besten beschreibt. Der Bresenham Algorithmus findet heutzutage vielfach Anwendung im Bereich der Computergrafik. Hier liegt eine Diskretisierung durch die Auflösung des Computers in Pixeln vor. Mittels des Bresenham Algorithmus kann bestimmt werden, durch welche Pixel eine Linie oder ein Liniensegment beschrieben werden kann. Abbildung 4.4 zeigt die initiale Idee des Bresenham Algorithmus in zwei Dimensionen. Eine ähnliche Diskretisierung weist auch die in dieser Arbeit verwendete TSDF-Karte auf. Sie ist allerdings im Gegensatz zu den beschriebenen Beispielen in drei Dimensionen diskretisiert. Mittels Bresenham soll bestimmt werden, welche Voxel der TSDF-Karte zu einem Ray gehören, der von einer Position  $\vec{p}$  ausgesendet wurde und sich mit der TSDF-Karte schneidet. Ziel ist die Beschleunigung des Ray-Tracing Ansatzes durch die Ausnutzung der Registrierung der Karte. Dabei gelten die gleichen Voraussetzungen wie beim zuvor beschriebenen Ray-Tracing und die Initialisierung der einzelnen Rays erfolgt analog. Im Gegensatz zum Ray-Tracing wird allerdings nicht der Ray schrittweise verlängert, sondern basierend auf den initialen Richtungsvektoren der Rays jeweils das nächste Voxel berechnet, das den Ray am besten beschreibt.

Die Grundlage für die Berechnung der zum Ray gehörigen Voxel bildet ein Startvoxel  $V_{start}^{\rightarrow}$ , gegeben durch die aktuell betrachtete Pose beziehungsweise Scannerposition  $P_i$  und ein Endvoxel  $V_{end}^{\rightarrow}$ . Letzterer berechnet sich aus dem Schnittpunkt des betrachteten Rays mit der Bounding-Box der lokalen Karte, gekennzeichnet durch ihren Ursprung  $plmap$  und ihre Seitenlängen  $(s_x, s_y, s_z)^T \dots$ . Basierend auf den berechneten Start- und Endvoxeln jedes Rays können nun die dazwischenliegenden Voxel mittels Bresenham ermittelt werden. Der Pseudo-Code in Abbildung

ggf. Mathematik dahinter erklären

1 legt die Logik für die Bestimmung der der Voxel dar, die die Linie gegeben durch den Startvoxel  $\vec{V}_{start}$  und Endvoxel  $\vec{V}_{end}$  beschrieben.

---

**Algorithm 1** Bresenham Algorithmus adaptiert in 3D (nach [5])

---

```

1: procedure BRESENHAM(  $\vec{V}_{start}, map_l, r_i^j$  )
2:   Initialisierung:
3:     Berechne die Endposition  $\vec{V}_{end}$  als Schnittpunkt des Rays  $r_i^j$  mit der lokalen Karte  $map_l$ 
4:     Berechne Bresenham Parameter (absolute Abstände und Raumrichtungen):
5:      $dx = \left| \vec{V}_{end}^x - \vec{V}_{start}^x \right|$  // Absolute zwischen den Punkten in alle Raumrichtungen
6:      $dy = \left| \vec{V}_{end}^y - \vec{V}_{start}^y \right|$ 
7:      $dz = \left| \vec{V}_{end}^z - \vec{V}_{start}^z \right|$ 
8:      $dm = \max(dx, dy, dz)$  // Maximale Komponente der Manhattan Distanz
9:      $sx = \vec{V}_{start}^x < \vec{V}_{end}^x ? 1 : -1$  // Raumrichtung
10:     $sy = \vec{V}_{start}^y < \vec{V}_{end}^y ? 1 : -1$ 
11:     $sz = \vec{V}_{start}^z < \vec{V}_{end}^z ? 1 : -1$ 
12:   Temporäre Vektoren zur Iteration initialisieren:
13:    $\vec{v}_0 = (x_0, y_0, z_0)^T = \vec{V}_{start}$  und  $\vec{v}_0 = (x_1, y_1, z_1)^T = \left( \frac{dm}{2}, \frac{dm}{2}, \frac{dm}{2} \right)^T$ 
14:   for  $i = 1; i < dm; i++ \text{ do}$  // ( $dm - 1$ ) mal iterieren
15:     Berechnung des nächsten Voxels (gegeben durch  $\vec{v}_0$ )
16:      $x_1 = x_1 - dx;$  if ( $x_1 < 0$ ) { $x_1 += dm;$   $x_0 += s_x;$ }
17:      $y_1 = y_1 - dy;$  if ( $y_1 < 0$ ) { $y_1 += dm;$   $y_0 += s_y;$ }
18:      $z_1 = z_1 - dz;$  if ( $z_1 < 0$ ) { $z_1 += dm;$   $z_0 += s_z;$ }
19:     Neues Linien-Voxel gegeben durch:  $\vec{v}_0$  bzw.  $(x_0, y_0, z_0)^T$ 
20:   end for
21: end procedure

```

---

Mit Hilfe dieser Logik lässt sich iterativ für jeden Ray  $r_i^j \in R_i$  der nächste zugehörige Voxel berechnen und evaluieren. Die Evaluation der von Bresenham ermittelten, aktuell vom Ray getroffenen TSDF-Zellen erfolgt analog zu der Evaluation des Ray-Tracing entsprechend des Zustandsdiagramms in Abbildung 4.3. Das Abbruchkriterium für jeden Ray  $r_i^j$  beim Bresenham ist entweder das Erreichen des Zielzustands im Zustandsdiagramm oder das Erreichen des zu  $r_i^j$  zugehörigen Endvoxels  $\vec{V}_{end}$ . Insgesamt ergibt sich der in Abbildung 2 dargestellte Pseudo-Code, bei variabler Verwendung von Bresenham oder Ray-Tracing. Dieser Pseudo Code wird zur Assoziationsbestimmung ausgehend von jeder Pose  $P_i$  ausgeführt, für die die Assoziationen bestimmt werden sollen.

---

**Algorithm 2** Asssoziations-Berechnung mittels Bresenham oder Ray-Tracing

---

```

1: procedure ASSOZIATIONS-BERECHNUNG(  $P_i, map_{local}$  )
2:   Initialization:
3:     Initialisiere die Rays  $R_i$  um  $P_i$  wie beschrieben in 4.3.1
4:     Initialisiere ein Array  $r_{status}$  der Größe der Anzahl von Rays, welches die aktuellen
   Zustände der Rays, gegeben durch das Zustandsdiagramm in 4.3 beschreibt
5:     Initialisiere einen Zähler für die Anzahl der fertigen Rays:  $cnt_{fin} = 0$ 
6:     while  $cnt_{fin} < size(r_{status})$  do
7:       for  $r_i^j$  in  $R_i$  do
8:         Ermittle nächstes Voxel  $V_i^j$  mittels Ray-Tracing oder Bresenham
9:         Evaluiere  $V_i^j$  auf Basis des Zustandsdiagramms in 4.3
10:        Speichere  $V_i^j$  als Assoziation, wenn es gemäß 4.3 mit  $P_i$  assoziiert ist
11:       end for
12:     end while
13:     Speichere die gefundenen Assoziationen gemäß 4.1 in der HDF5
14:   end procedure

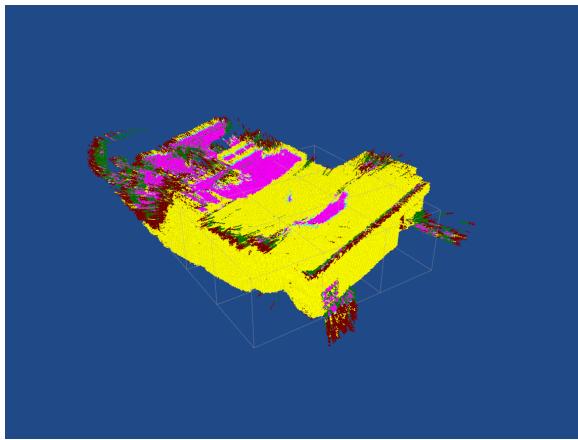
```

---

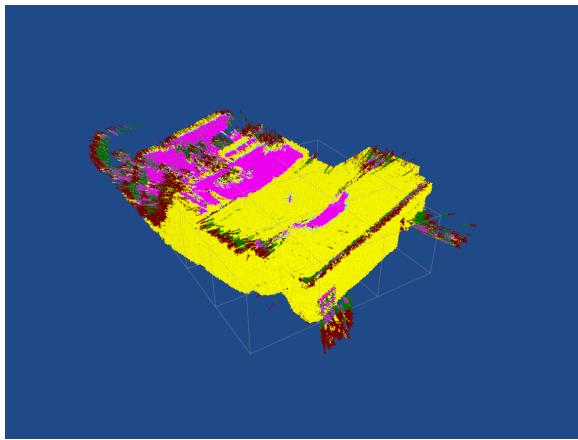
In den folgenden beiden Sktionen werden die Ergebnisse der Assoziationsbestimmung evaluiert und es findet ein Vergleich zwischen den beiden vorgestellten Algorithmen statt.

### 4.3.3 Ergebnisse

Abbildung 4.5 zeigt eine Gegenüberstellung der Ergebnisse der Evaluationsbestimmung zwischen Bresenham und dem Ray-Tracing Ansatz. Es ist ersichtlich, dass in beiden Figuren sehr ähnliche Ergebnisse erzielt werden konnten. Dies lässt sich auch an dem Prozentsatz der assoziierten Zellen von der Gesamtheit der Zellen kenntlich machen. Hier beträgt der Unterschied zwischen den beiden Ansätzen lediglich 0.23%, die vom Ray-Tracer zusätzlich assoziiert wurden. Dieser Unterschied lässt sich anhand einiger Eigenschaften von Bresenham deutlich machen. Der Algorithmus erlaubt unter gewissen Umständen, dass aufeinander folgende Zellen nur an der Ecke miteinander verbunden sind. Dann gilt zum Beispiel  $C_i = (0, 0, 0)^T$  und  $C_{i+1} = (1, 1, 1)^T$ . Dies ist eine der Ursachen die dazu führt, dass grundsätzlich nicht alle Zellen assoziiert werden können. Auch das Ray-Tracing weist ähnliche Probleme aufgrund der Diskretisierung der Schritte auf. Hier können durch die diskreten Schritte Zellen übersprungen werden, die in einer kontinuierlichen Betrachtungsweise vom Ray getroffen werden. Dieses Problem ist beim Ray-Tracer allerdings vernachlässigbar, solange die Schrittweite identisch zu derer gewählt wird, die beim Update der TSDF-Karte, in welchem nach Kapitel ?? ebenfalls ein Ray-Tracing Ansatz gewählt wurde, verwendet wird. Dadurch kann sichergestellt werden, dass Zellen, die beim Update der TSDF-Karte ausgehend von  $P_i$  verändert wurden, auch von dem Ray-Tracer zur Datenassoziation getroffen werden. Dies erklärt dementsprechend nicht die restlichen knapp 9% der Zellen, die durch den Ray-Tracer nicht assoziiert werden konnten. Der Prozentsatz variiert je nach verwendetem Datensatz und der Struktur der Daten. Nachfolgender Abschnitt erörtert das Problem der nicht assoziierten Zellen.



(a) Assoziationen identifiziert mit Bresenham.



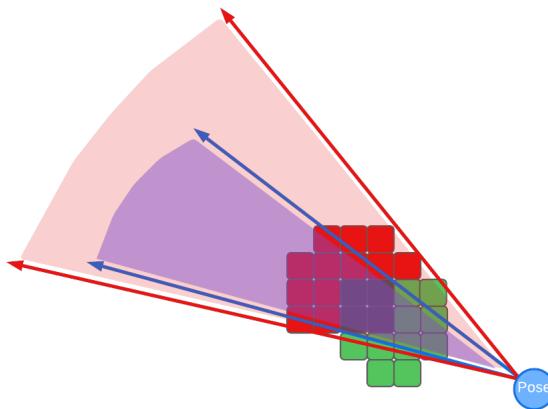
(b) Assoziationen identifiziert mit Ray-Tracing.

**Abbildung 4.5:** Gegenüberstellung der generierten Assoziationen für einen Beispieldatensatz. Die durch Bresenham gefundenen Assoziationen sind auf der linken Seite, die durch Ray-Tracing auf der rechten dargestellt. In diesem Fall wurden von Bresenham 91,74 Prozent der Zellen assoziiert, während vom Ray-Tracing 91,97 Prozent der Zellen assoziiert wurden. Zellen in gelb: assoziierte TSDF Zellen mit  $value < 0$ , Zellen in pink: assoziierte TSDF Zellen mit  $value > 0$ , Zellen in türkis: approximierter Nulldurchgang (Wechsel von positivem zu negativem Wert), Rest: nicht assoziierte Zellen. Zellen die nicht assoziiert werden, sind in diesem Fall größtenteils Teil von Verdeckungen oder Reflexionen des Laserscans und können als Outlier identifiziert werden.

#### 4.3.4 Evaluation

Diese Sektion befasst sich mit den Problematiken bei der Bestimmung von Assoziationen zwischen einer gegebenen Trajektorie und einer bestehenden TSDF-Karte. Zusätzlich erfolgt im zweiten Teil eine Evaluation der Laufzeiten zwischen dem Bresenham Algorithmus und dem Ray-Tracing.

Im vorigen Abschnitt wurde beschrieben, dass ein je nach Datensatz variabler Prozentsatz der Karte nicht durch den Ray-Tracer und ebenfalls nicht durch Bresenham mit einer der Posen assoziiert werden kann. Dieses Problem wird sowohl durch die diskreten Schritte beim Ray-Tracer, als auch durch eine unpassende Wahl der nächsten Zelle im 3D Bresenham Algorithmus ausgelöst. Diese beiden Probleme sorgen allerdings nur zum Teil für die nicht assoziierten Zellen. Eine wesentliche Ursache sind durch die Diskretisierung eintretende Verdeckungen und Reflexionen beziehungsweise nicht gefiltertes Sensorrauschen bei der Generation der Karte, das für einzelne in der Luft schwebende TSDF-Zellen sorgt. Ein Beispiel für beschriebene Verdeckungen sind zum Beispiel Türen. In der Realität kann ein Laserscanner beliebig nah am Türrahmen vorbei scannen. Hier wird der Sichtbereich durch die Tür lediglich vom Türrahmen eingeschränkt. In einer diskretisierten TSDF-Karte wird der Sichtbereich durch die Tür je nach der Auflösung der Karte eingeschränkt. Abbildung ?? zeigt dieses Problem schematisch in "D anhand einer Pose und einem Hindernis, wie zum Beispiel einem Pfeiler. Es ist erkenntlich, dass ein Teil des ursprünglichen Sichtbereichs nun durch die diskretisierte TSDF-Karte verdeckt ist und alle Zellen, die sich im verdeckten Bereich hinter dem Hindernis nicht mehr assoziiert werden können. Dies ist die Hauptursache für beschriebene Probleme bei der Assoziationsbestimmung und kann ohne



**Abbildung 4.6:** Diese Abbildung zeigt die Vergrößerung des Verdeckungsbereichs durch die Diskretisierung der Karte als TSDF. Zu sehen ist der ehemalige Verdeckungsbereich des in schwarz gefärbten Objektes. Dieser ist als halbtransparenter blauer Bereich gekennzeichnet. Darum herum zeigt sich der neue Verdeckungsbereich, der durch die Diskretisierung der Karte auftritt, ausgehend von der aktuell betrachteten Pose. Er ist halbtransparent in rot dargestellt. Diese Vergrößerung kommt durch die Beschaffenheit der Karte zustande. Nur außerhalb der roten Pfeile, also außerhalb des Verdeckungsbereichs kann mit Sicherheit gesagt werden, dass der künstliche Ray Freiraum passiert. Durch die Diskretisierung hat sich der Bereich, in dem ein Objekt sein könnte, vergrößert. Dies ist die Hauptursache für nicht assoziierbare Zellen.

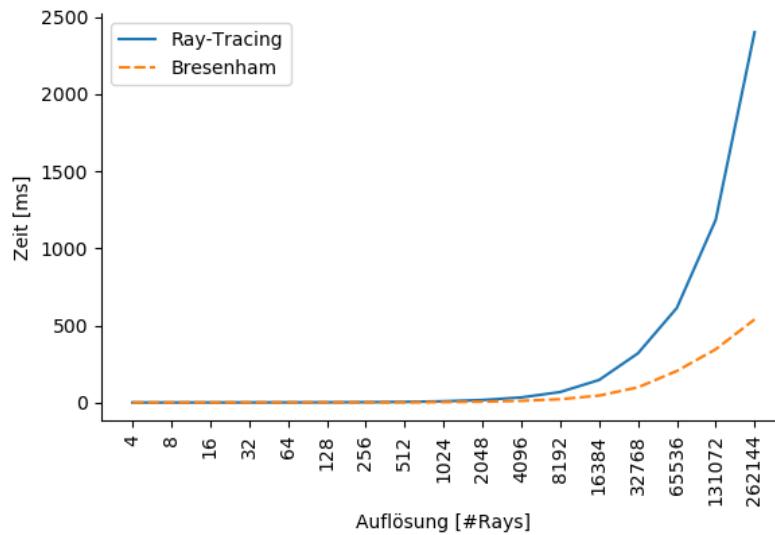
zusätzliche Informationen innerhalb der TSDF-Karte nicht gelöst werden. Eine Möglichkeit wäre, es dem Ray-Tracer zu erlauben, durch Wände hindurch zu sehen, was allerdings dazu führen könnte, dass Zellen mit der Pose assoziiert werden, die nicht zu ihr gehören. Dies ist einer der Hauptgründe für eine grundlegende Veränderung der Herangehensweise an das in dieser Arbeit formulierte Problem. 4.4 eruiert weitere Probleme dieses ersten Ansatzes.

Abbildung 4.7 zeigte eine Gegenüberstellung der Laufzeiten des Bresenham-Algorithmus in 3D gegenüber des vorgestellten Ray-Tracing Ansatzes. Es wird deutlich, dass durch die Einführung des Bresenham Algorithmus eine deutliche Steigerung der Effizienz erzielt wird.

Der folgende Absatz beschäftigt sich mit der Identifikation von Loop Closures innerhalb dieses ersten Ansatzes.

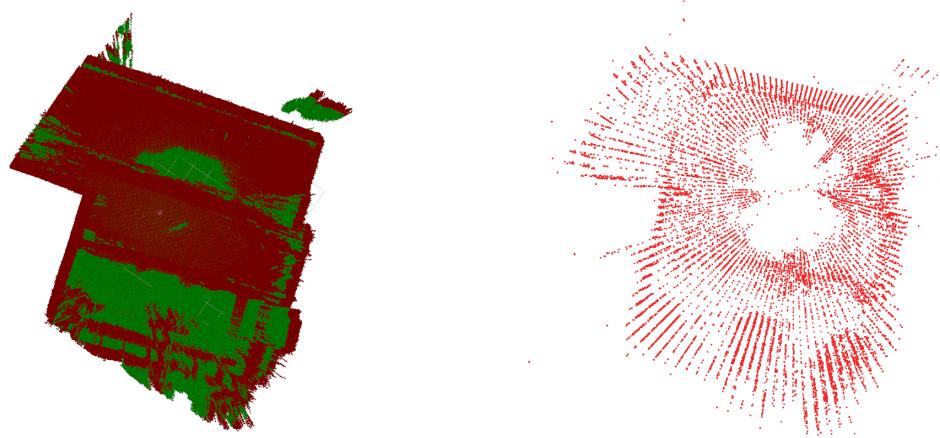
## 4.4 Loop Closure

Wie in 3.4 eingeführt wurde, ist es für die Identifikation von Schleifenschlüssen notwendig zu detektieren, dass der Roboters oder das Systems räumlich gesehen an derselben Pose oder in der Nähe der aktuellen Pose bereits gewesen ist. Dies lässt sich zum Beispiel über eine Distanzheuristik ermitteln. Alle Posen, deren euklidische Distanz zur aktuellen Pose geringer ist als eine festgelegte Schwelle sind Kandidaten für potentielle Schleifenschlüsse, die dann in den Posegraphen, gegeben durch die Posen des Roboters und gegebenenfalls vorige Schleifenschlüsse, eingefügt werden können. Um zu bestimmten, ob einer der ermittelten Schleifenschluss-Kandidaten für einen Schleifenschluss in Frage kommt, muss die Umgebung jeder einzelnen in Frage kommenden Pose



**Abbildung 4.7:** Laufzeiten der vorgestellten Algorithmen zur Bestimmung der Assoziationen. Es handelt sich bei den jeweiligen Werte um das Mittel der Laufzeiten mehrerer künstlicher Scans beider Algorithmen, ausgehend von ungefähr 30 verschiedenen Positionen. Es ist zu sehen, dass - obgleich der Zeitbedarf beider Algorithmen exponentiell mit der Anzahl an Rays steigt - der Bresenham Algorithmus wesentlich effizienter ist. Bei den Laufzeiten handelt es sich um nicht beschleunigte, rein CPU basierte Ansätze. Beide Varianten können durch die Nutzung einer **Graphics Processing Unit (GPU)** oder mittels CPU basierter Beschleunigung durch eine Parallelisierung deutlich beschleunigt werden. Diese Beschleunigung liegt jedoch nicht im Fokus dieser Arbeit. Dieses Benchmark bestätigt die Hypothese, dass die Laufzeit durch die Ausnutzung der Diskretisierung erheblich verbessert wird. wobei ähnliche Ergebnisse erzielt werden können.

gegen die Umgebung der aktuellen Pose verglichen werden. Das Mittel der Wahl ist hier ein Scan-Matching Ansatz wie vorgeschlagen in [4, 16, 23]. Dieser Ansatz basiert allerdings auf räumlichen Punktdaten, die an dieser Stelle nicht vorhanden sind. Die Einzige Repräsentation der Umgebung ist die bereits fertiggestellte TSDF-Kartendarstellung. Aus dieser können allerdings Punktwolken approximiert werden. Grundlage dafür bildet in dieser Arbeit der zuvor entwickelte Ray-Tracer. Dieser wird wie gehabt mit einer beliebigen Auflösung initialisiert und alle Rays werden schrittweise verlängert. Anstelle nun Assoziationen zu ermitteln, wird der erste Schnitt des Ray-Tracers mit einem Nulldurchgang, einem Wechsel von positiven auf negative TSDF-Werte, gesucht. Ist ein solcher gefunden, wird die Ebene zwischen den beiden Voxeln berechnet, die die gesuchte Oberfläche am besten beschreibt. Diese Ebene ergibt sich aus den Positionen der Voxel  $\vec{V}_i$  und  $\vec{V}_j$ , sowie den zugehörigen TSDF-Werten  $v_i$  und  $v_j$ . Die TSDF-Werte bestimmen, wo die Ebene zwischen den beiden Voxeln liegt. Gegeben die Zentren der Voxel  $\vec{c}_i$  und  $\vec{c}_j$  in globalen Koordinaten, sowie die zugehörigen TSDF-Werte ist die Ebene beschrieben durch einen Ortsvektor  $\vec{v}_{ij}$ , sowie eine Normale  $n_{ij}$ . Die beiden Komponenten berechnen sich wie folgt:



(a) Lokale TSDF Karte ausgehend von einer festgelegten Pose.

(b) Aus der lokalen Karte generierte Punktwolke.

**Abbildung 4.8:** Gegenüberstellung eines Ausschnitts der TSDF-Karte und einer dazugehörigen, approximierten Punktwolke.

$$\vec{v}_{ij} = \vec{c}_i + (\vec{c}_j - \vec{c}_i) \cdot \frac{|\vec{v}_i|}{|\vec{v}_i| + |\vec{v}_j|} \quad (4.9)$$

$$\vec{n}_{ij} = (\vec{c}_j - \vec{c}_i) \quad (4.10)$$

Auf Basis der durch  $\vec{v}_{ij}$  und  $\vec{n}_{ij}$  gegebenen Ebene lässt sich nun ein Punkt der Punktwolke als Schnitt des betrachteten Rays mit der zuvor berechneten Ebene errechnen. Das Ergebnis dieser Approximation ist in Abbildung 4.8 dargestellt. Eine Zuordnung der approximierten Punktwolke zur Pose von welcher ausgehend die Punktwolke generiert wurde ist jedoch nicht sinnvoll, da unbekannt ist, welcher Teil der Karte tatsächlich von dieser Pose aus generiert wurde. Dasselbe Problem gilt auch für die Bestimmung von Punktwolken aus einer statischen Karte zur Berechnung potentieller Schleifenschlüsse um auf Basis der Schleifenschlüsse erneut die Karte zu optimieren. Dies scheitert schon bei der Generierung von Punktwolken aus einer potentiell unvollständigen oder gänzlich falschen Karte. Die approximierten Punktwolken stellen dann in jedem Fall auch unvollständige oder falsche Daten dar. Auch ein Scan-Matching mit potentiell unvollständigen oder falschen Daten ist nicht zielführend. Aus den genannten Gründen wurde die Idee der Punktwolkenapproximation zur Identifikation von Schleifenschlüssen verworfen.

Im nachfolgenden Abschnitt wird auf ein mögliches Kartenumupdate unter der Prämisse eines zuvor identifizierten Schleifenschlusses eingegangen und die Ergebnisse eruiert.

## 4.5 Kartenupdate

Dieser Abschnitt befasst sich mit dem Problem des TSDF-Kartenupdates, gegeben ein initialer Pfad  $\mathfrak{P}_{init}$  und ein durch Schleifenschlüsse optimierter Pfad  $\mathfrak{P}_{opt}$ , sowie eine generierte Karte auf Basis des initialen Pfades. Ziel ist die Erstellung einer optimierten Karte, angepasst an den optimierten Pfad  $\mathfrak{P}_{opt}$ . Um dieses Ziel zu erreichen müssen Teilbereiche der Karte entsprechend der Veränderungen der Posen verschoben werden. Dazu ist eine Assoziation zwischen Teilen der Karte und den zugehörigen Posen herzustellen. Dieses Problem wurde in den vorigen Abschnitten bereits erörtert. Entsprechend wurde ein Manager implementiert, der die zu bestimmenden Assoziationen verwaltet. Jeder Pose des alten Pfades wird ein HDF5-serialisierbares Assoziations-Objekt zugewiesen. Die Assoziationen selbst werden in einer **1:N** Beziehung durch Ray-Tracing oder den vorgestellten 3D-Bresenham Ansatz generiert. Dies liefert für jede Position  $P_i \in \mathfrak{P}_{init}$  eine Menge  $M = \{C_1, \dots, C_N\}$ , wobei die Anzahl der Assoziationen für jede Position unterschiedlich ist.

Die grundsätzliche entwickelte Logik für das Update der Karte auf Basis der generierten Assoziationen wird im Folgenden diskutiert. Zunächst wird in einem ersten Schritt für jedes Pose-Paar aus initialem und korrigiertem Pfad eine Pose-Differenz errechnet. Diese wird später genutzt um die neue Position assoziierter Zellen zu bestimmen. Gegeben die initialen Posen aus Pfad  $\mathfrak{P}_{init}$ , sowie die optimierten Posen aus  $\mathfrak{P}_{opt}$ , berechnen sich die einzelnen Pose-Differenzen beziehungsweise Transformationen  $T_i$  wie folgt:

$$T_i = \left( T_{i \rightarrow map}^{opt} \right)^{-1} \cdot T_{i \rightarrow map}^{init} \quad (4.11)$$

Im Anschluss daran werden drei Level von Daten generiert, die jeweils aufeinander aufbauen und den Informationsgehalt erhöhen. Zwei dieser Level verwenden Hashmaps um schnell detektieren zu können ob ein Eintrag für eine Zellposition schon existiert um dann entsprechend reagieren zu können. **Level 1** enthält für jede assozierte TSDF-Zelle ein Tupel der alten Zellposition, der neuen, akkumulierten Zellposition, des zugehörigen TSDF-Werts und Gewichts, sowie einen Zähler für die Anzahl an Posen, die mit dieser Zelle assoziiert werden. Gegeben ein Array der mit der aktuellen Zelle  $\vec{C}_i$  assoziierten Posen  $[P_x, \dots, P_{x+n}]$ , sowie ein Array aus Pose-Differenzen, gegeben als Transformationsmatrix,  $[T_0, \dots, T_{N-1}]$  berechnet sich die akkumulierte Zellposition dabei wie folgt:

$$\vec{C}_{acc} = \sum_{j=x}^{x+n} T_j \cdot \vec{C}_i \quad (4.12)$$

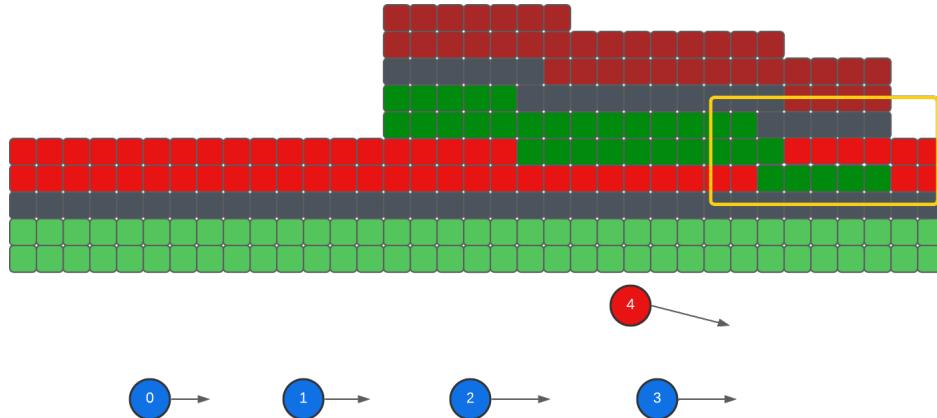
Anhand der generierten **Level 1** Daten lassen sich nun **Level 2** Daten generieren. Hier wird berücksichtigt, dass die neue Position mehrerer initialer TSDF-Zellen identisch sein kann. In diesem Fall wird hier zunächst das arithmetische Mittel der TSDF-Werte und Gewichte gebildet und der neuen Position zugewiesen. Da für die Akkumulation dieser Daten Hashmaps verwendet wurden, haben die nun generierten Daten keinerlei räumliche Zusammengehörigkeit. Dies führt zu Problemen, wenn die zu schreibende Zelle außerhalb der aktuellen lokalen Karte befindlich ist und diese entsprechend verschoben werden muss. Das Verschieben der Karte ist eine

Ressourcen und Zeit intensive Operation, die möglichst selten ausgeführt werden sollte. Durch die Unordnung muss im schlimmsten Fall für jede neue Zelle eine Verschiebung beziehungsweise ein **Shift** stattfinden. Bei Millionen von TSDF-Zellen und einer durchschnittlichen Dauer des Shifts von über einer Sekunde, würde alleine dieser Schritt über 10 Tage in Anspruch nehmen. Dementsprechend sind **Level 3** Daten spezifisch angeordnet, um die Anzahl an Shifts auf ein Minimum zu reduzieren. Dazu wird die **3D Bounding Box** des Raumes berechnet, den die neuen Zellpositionen einnehmen. Durch eine Aufteilung der Bounding-Box in Quader der Größe der lokalen Karte und eine Zuordnung der Zellen zu jedem dieser Quader wird die Anzahl der Shifts auf das gesuchte Minimum reduziert und die benötigte Zeit von Tagen auf Sekunden reduziert. Schlussendlich werden die initialen, assoziierten Zellen gelöscht und die neuen Zellpositionen mit den berechneten TSDF-Werten und Gewichten in die Karte geschrieben. Alle nicht assoziierbaren Zellen werden gelöscht. Da allerdings in der Regel kein globales, sondern nur ein lokales Update der TSDF-Karte vorgenommen werden soll lässt sich durch eine lokale Assoziationsbestimmung nicht sagen, ob die nicht assoziierbaren Zellen bedenkenlos gelöscht werden können, weil sie gegebenenfalls in einer globalen Assoziationsbestimmung assoziiert werden könnten. Ohne sicherzustellen, dass dies nicht der Fall ist, riskiert ein Löschen der genannten Zellen einen nicht zu kompensierenden Informationsverlust, was die Idee eines lokalen Updates in diesem Falle zunichte macht.

Wie in vorigen Abschnitten herausgestellt ist, kann zusätzlich nicht mit Sicherheit gesagt werden, dass jede so assoziierte Zelle auch von der assoziierten Pose generiert wurde. Dies lässt sich anhand von Abbildung 4.9 eruieren. Durch einen Fehler in der Registrierung im initialen Pfad befindet sich eine Pose an einer fehlerhaften Stelle. Von dieser Pose aus wurde allerdings bereits ein - ebenfalls fehlerhaftes - TSDF Update durchgeführt. Die Daten, die zu dieser Pose gehören befinden sich nun hinter einer Wand, die zuvor von anderen Posen aus erstellt wurde. An dieser Stelle kann nun unmöglich ermittelt werden, welche Pose für die Erstellung der fehlerhaften Wand verantwortlich ist. Da der implementierte Ray-Tracer nicht durch Wände hindurch schauen kann, wird daher ein falscher Teilbereich der Karte mit dieser fehlerhaften Pose assoziiert. Das Problem doppelter Wände ist dabei ein Problem, dass durch akkumulierten Drift im SLAM regelmäßig auftritt und durch Schleifenschlüsse gelöst werden kann. Dazu müssen die fehlerhaften Daten, wie zum Beispiel fehlerhaft positionierte Wände, allerdings auch mit den entsprechenden Posen assoziiert werden können. Diese Prämisse kann mit diesem Ansatz nicht erfüllt werden.

Zusätzlich zum oben beschriebenen Problem existiert ein weiteres Problem, dass die Zellennachbarschaften betrifft. Benachbarte Zellen einer Wand sollten auch in der korrigierten Variante in direkter Nachbarschaft liegen, sodass die lokale Konsistenz der Karte gewährt bleibt. Wird nun eine Zelle von zwei verschiedenen Pose assoziiert und auf Basis eines Mittels der Pose-Änderungen verschoben, die Nachbarzelle allerdings durch einen Diskretisierungsfehler nur von einer Pose aus assoziiert und entsprechend verschoben, werden die beiden Zellen auseinander gezogen. Dies sorgt für ein merkliches Rauschen in der Karte und für zusätzliche Ungenauigkeiten. Eine Lösung hier wäre die lokale Konnektivität der Karte beim Update zu berücksichtigen. Dies wurde im Rahmen dieser Arbeit, besonders aufgrund der oben genannten Probleme allerdings nicht weiter verfolgt.

Um den oben beschriebenen Update-Prozess zu evaluieren, wurden einfache Transformationen, wie



**Abbildung 4.9:** TSDF-Karte nach Update durch eine fehlerhaft registrierte Pose (Pose 4, dargestellt in rot). Der fehlerhaft hinzugefügte Teil der TSDF-Karte ist farblich durch einen dunkleren Rot- und Grünton hervorgehoben. Eine Assoziationsbestimmung ausgehend von Pose 4 würde hier nicht den fehlerhaften Teil der Karte assoziieren da dieser hinter einer Wand befindlich ist. Der stattdessen mit dieser Pose assoziierte Teil der Karte wurde ursprünglich von anderen Positionen aus generiert. Der gelbe Rahmen markiert einen Teil der Karte, indem bereits generierte TSDF-Zellen mit den Werten neu erstellter Zellen verrechnet werden müssen. Der neue TSDF-Wert errechnet sich dann gemäß der alten, sowie neuen Gewichte und Werte. Ein mögliches Szenario dieses Updates für den Fall, dass die Gewichte beider Zellen jeweils 1 ist, ist in der Abbildung dargestellt. Es wird deutlich, dass die TSDF-Karte durch die fehlerhaften Daten nun eine Inkonsistenz in den Gradienten aufweist, die nicht ohne Weiteres aufgelöst werden kann. Die Farben der Zellen im Überlappungsbereich wurde auf Basis der Zelle gewählt, die dominiert, also den größeren Einfluss auf das Endergebnis hat.

eine gleichmäßige Translation oder Rotation des Pfades vorgenommen und ein Kartenupdate auf Basis dieses neuen Pfades durchgeführt. Gleichmäßige Translationen stellten in dieser Hinsicht mit Ausnahme der nicht assoziierbaren Zellen keine Probleme dar. Sobald der Pfad oder Teile des Pfades in eine beliebige Richtung rotiert werden, trifft dies nicht mehr zu und durch die beschriebenen Probleme verliert die Karte ihre lokale Konnektivität und es bleibt lediglich ein inkonsistentes Rauschen. Fast jeglicher räumlicher Informationsgehalt geht verloren.

Nachfolgender Abschnitt evaluiert die um jetzigen Zeitpunkt gefundenen Erkenntnisse und beschreibt, wie im weiteren Verlauf der Arbeit vorgegangen wird.

## 4.6 Evaluation

Ziel dieses Kapitels ist die die Evaluation des Ansatzes zur Optimierung eines gegebenen Pfades durch Schleifenschlüsse und - damit verbunden - eine Optimierung der initial gegebenen TSDF-Karte auf Basis der zugrunde liegenden Pose-Änderungen. Die vorigen Absätze haben gezeigt, dass in grundlegenden Fällen, wie gleichmäßigen Translationen, ein Update mit dem vorgestellten Ansatz möglich ist. Diese grundlegende Fälle sind allerdings eine Abstraktion, die im realen Fall nicht gegeben ist. Zusätzlich scheitert der Ansatz bereits an der Ermittlung des optimierten

Pfades. Wie oben beschrieben ist das Auffinden von Schleifenschlüssen und damit verbunden eine Optimierung des Pfades basierend auf den gegebenen Daten nicht möglich. Zudem gibt es mehrere Probleme bei der Ermittlung der Assoziationen, wie vergrößerte Verdeckungsbereiche durch die Diskretisierung der TSDF-Karte oder gänzlich falsch eingetragene Teilbereiche der Karte, die wiederum von anderen Teilen der Karte verdeckt und entsprechend nicht mehr assoziiert werden können. Aufgrund der Fülle der genannten Probleme und dem Nichtvorhandensein von Lösungen wird nun an dieser Stelle der erste Ansatz verworfen. Stattdessen wird im Folgenden auf einer größeren Datenbasis gearbeitet, die auch die Nutzung aufgenommener Punktwolken erlaubt. Dies ermöglicht eine Nutzung dieses Ansatzes sowohl innerhalb eines vorhandenen SLAM Ansatzes, als auch losgelöst in einem Nachbearbeitungsschritt. Ziel ist die Untersuchung von Schleifenschlüssen, sowie globalen und partiellen Updates der TSDF-Karte auf Basis dieser erweiterten Datenbasis. Das folgende Kapitel widmet sich nun der Grundlage für die Optimierung der Karte, der Identifikation von Schleifenschlüssen und damit verbunden der Optimierung der Roboter-Trajektorie beziehungsweise des Pfades.

# Kapitel 5

## Schleifenschlüsse

Dieses Kapitel beschäftigt sich mit der Detektion von Schleifenschlüssen und der damit verbundenen Optimierung der initialen Schätzung einer Trajektorie beziehungsweise eines Pose-Graphen, entlang dessen ein Sensorsystem bewegt wurde. Im Folgenden wird zunächst näher auf den Detektionsschritt eingegangen. Auf Basis dessen wird in den drauf folgenden Abschnitten die in dieser Arbeit verwendete Methode zur Optimierung des Pose-Graphen eruiert.

### 5.1 Detektion von Schleifenschlüssen

Dieser Abschnitt befasst sich mit der Detektion von Schleifenschlüssen in einem Pose-Graphen. Die hier vorgestellte Methode kann sowohl in einem inkrementell erweiterten Pose-Graphen, zum Beispiel als zusätzlicher Schritt in einem SLAM Verfahren, als auch in einem Nachbearbeitungsschritt auf einen bereits fertigen Graphen angewandt werden. Die Methode und die Optimierung des Graphen ist dabei zunächst vollständig losgelöst von der TSDF-Karte, die im Anschluss optimiert wird. Mehr dazu in Kapitel 6. Die vorliegenden initiale Schätzung der Roboter-Trajektorie kann das Ergebnis eines inkrementellen SLAM Ansatzes sein oder lediglich auf Odometrie-Schätzung beruhen.

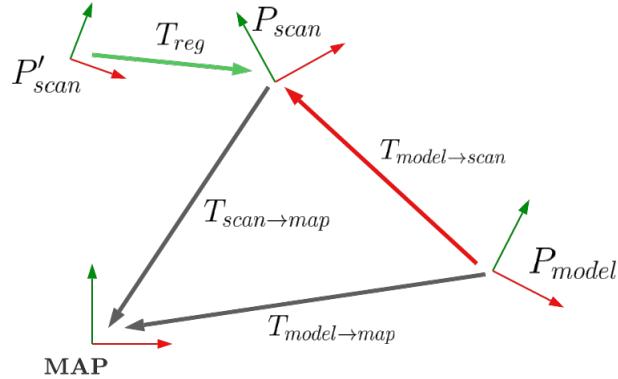
Bevor nach Schleifenschlüssen gesucht werden kann, gilt es zu bestimmen, von welcher Pose aus gesucht werden soll. In einem inkrementellen SLAM Verfahren wäre das jeweils die aktuell betrachtete, zuletzt in den Graphen eingefügte Pose. In einem Nachbearbeitungsschritt wird dieser inkrementelle Prozess imitiert, indem der Pose-Graph, beginnend von der ersten eingefügten Pose, abgelaufen wird. In beiden Fällen wird also ein Schleifenschluss mit Posen gesucht, die früher in den Graphen eingefügt wurden als die aktuelle Pose  $P_{cur}$ . In einem ersten Schritt wird hierzu eine Menge von Schleifenschluss-Kandidaten erstellt. Diese Menge ergibt sich aus der in [4, 23] vorgestellten euklidischen Distanzmetrik, die in dieser Arbeit verwendet wird. Eine Pose  $P_i$  gilt als Schleifenschluss-Kandidat zu  $P_{cur}$ , wenn die euklidische Distanz zwischen den beiden Posen geringer ist als eine parametrisierbare Schwelle  $d_{max}$ . Zusätzlich zu diesem Parameter wird an dieser Stelle ein weiterer Parameter  $d_{trav}$  eingeführt der die minimale Distanz definiert, die der Sensorsystem entlang des Teilstücks gegeben durch  $P_i$  und  $P_{cur}$  zurückgelegt hat. Um

Posen zu identifizieren, die diese Voraussetzungen Erfüllen, wird ein **k-d tree (k-d Baum)** [1] verwendet. Ein k-d Baum ist eine Datenstruktur zur Speicherung von (räumlichen) Informationen, die durch assoziative Suchen abgefragt werden können. Ein k-d Baum eignet sich aufgrund der optimierten Laufzeit besonders für räumliche Suchen. Dabei steht das **k** für die Dimensionalität des Suchraums [1]. An dieser Stelle wird der k-d Baum zur Speicherung von **3-d** Daten genutzt. Diese ergeben sich aus den Posen des Pfades. Vor jeder Detektion wird ein neuer k-d Baum aus den Translationsanteilen aller zuvor eingefügten Posen aufgebaut. Im Anschluss wird eine Radius-Suche durchgeführt, die alle Daten des k-d Baumes (hier 3-d Punkte) zurückliefert, deren euklidische Distanz zu einer übergebenen Position geringer ist als eine übergebene Schwelle. Die übergebene Position ist dabei der Translationsanteil von  $P_{cur}$  und die übergebene Schwelle  $d_{max}$ . Ergebnis ist eine Menge von Kandidaten für Schleifenschlüsse  $\mathbb{K} = \{K_0, \dots, K_k\}$ . Diese Arbeit verwendet die k-d Baum Implementation der **Pointcloud Library (PCL)** ??.

Über die generierten Kandidaten ist an dieser Stelle lediglich bekannt, dass sie sich im nicht optimierten Pfad in der Nähe der aktuellen Pose  $P_{cur}$  befinden. Diese räumliche Nähe ist an dieser Stelle allerdings nur eine Annahme, die es zu verifizieren gilt. Als Basis für die Verifikation dient die in diesem Ansatz gegebene Datenbasis in Form von zu jeder Pose des Pose-Graphen  $\mathfrak{P}_{init} = \{P_0, \dots, n\}$  zugehörigen Punktwolken  $\mathbb{C} = \{C_0, \dots, C_n\}$ . Für die aktuell betrachtete Pose  $P_{cur}$  und die zugehörige Punktfolge wird ein Scan-Matching gegen jeden Kandidaten aus  $\mathbb{K}$  und dessen zugehörigen Punktfolgen  $C_i$  durchgeführt und evaluiert. Konvergiert das Scan-Matching mit einer festgelegten Genauigkeit ist der Kandidat validiert. Im Folgenden wird die Punktfolge der aktuell betrachteten Pose  $P_{cur}$  als Scan-Punktfolge  $C_{scan}$  und die zum zu validierenden Kandidaten zugehörige Punktfolge als Model-Punktfolge  $C_{model}$  bezeichnet. Grundlage für das Scan-Matching der beiden Punktfolgen ist ein Algorithmus wie **ICP**, **GICP** oder vergleichbare Algorithmen zur Registrierung von Punktfolgen. Diese Algorithmen liefern neben einer Information zur Konvergenz in der Regel zusätzlich ein Maß dafür zurück, wie gut die Punktfolgen nach der Registrierung aufeinander passen. In dieser Arbeit werden hierzu die Implementationen der Algorithmen der Pointcloud Library [20] verwendet. Diese liefern für beide Algorithmen einen **Fitness-Score** ( $\Gamma$ ) zurück, der die durchschnittliche quadrierte Distanz zwischen einem Punkt der Scan-Punktfolge und dem euklidisch nächste Punkt der Model-Punktfolge nach Registrierung der Scan-Punktfolge an die Model-Punktfolge beschreibt. Nachfolgende Formel zeigt diesen Sachverhalt mit  $s_i$  als aktuellen Scanpunkt und  $m_i$  als zugehörigen, euklidisch nächsten Punkt der Model-Punktfolge und  $N$  als Anzahl der Punkte in der Scan-Punktfolge.

$$\Gamma = \frac{\sum_{i=0}^N \|\vec{m}_i - \vec{s}_i\|^2}{N} \quad (5.1)$$

Liegt der Fitness-Score unter einer vordefinierten Schwelle  $\Gamma_{max}$ , ist der Kandidat validiert. Zusätzlich liefern die Algorithmen die bestimmte finale Transformation  $T_{fin}$  der Registrierung von Scan zu Model. Da GICP und ICP deutlich bessere Ergebnisse erzielen, wenn zwischen den Daten bereits eine Initialschätzung vorliegt, wird eine solche, auf Basis der jeweiligen Pose-Differenzen, genutzt. Die Punktfolgendaten liegen hier jeweils relativ zur Pose, von derer sie aufgenommen wurden, vor. Für die Initialschätzung wird nun die Model-Punktfolge ins Koordinatensystem



**Abbildung 5.1:** Diese Grafik stellt die Posen und Transformationen dar, die bei der Detektion eines Schleifenschlusses zur Verwendung kommen. Dabei bestimmt  $T_{model \rightarrow scan}$  die Vortransformation vom lokalen Model-Koordinatensystem in das des Scans (dargestellt in rot). In grün dargestellt ist die Transformation  $T_{reg}$ , die von dem verwendeten Algorithmus zur Registrierung der Scan-Punktwolke an die mit  $T_{model \rightarrow scan}$  vortransformierte Model-Punktwolke, ausgegeben wird. Eine Kombination der Transformationen  $T_{reg}$  und  $T_{model \rightarrow scan}$  ergibt die finale, zur Optimierung des Pose-Graphen verwendete Transformationen. Diese Kombination ist in Gleichung 5.3 definiert.

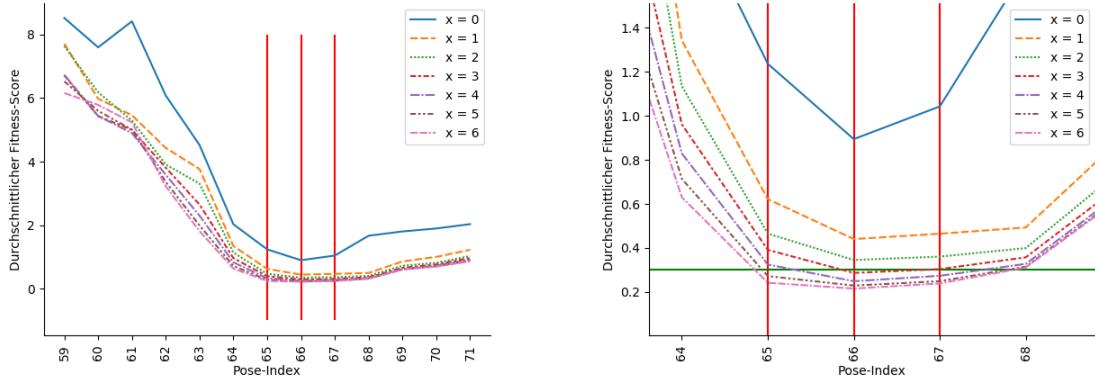
der Scan-Punktwolke transformiert. Dazu wird zunächst die Transformation  $T_{model \rightarrow scan}$  über die zugehörigen Posen  $P_{model}$  und  $P_{scan}$  und die zugehörigen Transformationen  $T_{model \rightarrow map}$  und  $T_{scan \rightarrow map}$  von den Posen ins Ursprungskoordinatensystem bestimmt:

$$T_{model \rightarrow scan} = T_{scan \rightarrow map}^{-1} \cdot T_{model \rightarrow map} \quad (5.2)$$

Im Anschluss wird jeder Punkt  $p_{model}^i$  der Punktwolke  $C_{model}$  mit der bestimmten Transformation  $T_{model \rightarrow scan}$  transformiert. Nun ist die Model-Punktwolke in das Koordinatensystem der Scan-Punktwolke, gegeben die aktuellen Pose-Schätzungen, vortransformiert. Diese Vortransformation muss nach der anschließenden Registrierung mittels ICP, GICP oder einem ähnlichen Verfahren für die Berechnung der finalen Transformation berücksichtigt werden. Dies ist in Abbildung 5.1 dargestellt. Sie führt zusätzlich die vom zur Registrierung berechneten Algorithmus bestimmte Transformation  $T_{reg}$ , sowie die sich daraus ergebene neue, approximative Pose  $P'_{scan}$ , von der die Scan-Punktwolke aufgenommen wurde. Die finale Transformation  $T_{scan' \rightarrow model}$  die im weiteren Verlauf für die Optimierung des Pose-Graphen benötigt wird ergibt sich wie folgt:

$$T_{scan' \rightarrow model} = T_{model \rightarrow scan}^{-1} \cdot T_{reg} \quad (5.3)$$

Auf diese Art und Weise können nur wenige Schleifenschlüsse identifiziert werden. Ursache ist zu diesem Zeitpunkt das Scan-Matching, welches zwar in den meisten Fällen konvergiert, jedoch fast keinen der Kandidaten aufgrund zu hoher Fitness-Scores validieren kann. Zusätzlich sind einige der identifizierten Schleifenschlüsse, besonders in Bereichen von Fluren oder Orten



(a) Durchschnittlicher Fitness-Score für gefundene Schleifenschluss-Kandidaten eines Datensatz von Pose 0 bis Pose 80, Posen ohne Kandidaten sind nicht dargestellt.

(b) Vergrößerung des Bereiches von links, in dem Schleifenschlüsse gefunden und validiert wurden. Dargestellt als grüne horizontale Linie ist die parametrisierte Schranke für Schleifenschlüsse.

**Abbildung 5.2:** Diese Grafik zeigt den Einfluss des Fensters für die Anreicherung der Model-Punktwolke bei der Validierung gefundener Schleifenschluss-Kandidaten, gegeben die Größe des Fensters  $x$ . Rote vertikale Linien markieren Posen, an den Schleifenschlüsse detektiert und aufgrund des geringen Fitness-Scores validiert wurden. Die Gesamtmenge der Posen, deren Punktwolken zu einer dichteren Model-Punktwolke zusammengeführt werden, beträgt  $2 \cdot x + 1$ . Es ist zu erkennen, dass der durchschnittlich berechnete Fitness-Score mit steigendem  $x$  im Schnitt niedriger ist. Allerdings wird die Verringerung ab einem gewissen Punkt unwesentlich. Hier sind zwischen einer Wahl von  $x = 2$  bis  $x = 6$  nur geringe Unterschiede zu erkennen. Die größten Veränderungen sind zwischen  $x = 0$  und  $x = 2$  zu erkennen. Hier sinkt der durchschnittliche Fitness-Score merklich. Dies trifft auch auf die Ergebnisse im Scan-Matching zu. Diese verbessern sich bei zunehmendem  $x$ , hier sind ebenfalls die größten Veränderungen zwischen  $x = 0$  und  $x = 2$  zu sehen. Die Wahl des Fenster korreliert zusätzlich mit dem durchschnittlichen absoluten Fehler zwischen der durch die Schleifenschlüsse optimierten Trajektorie eines Datensatz und der zugehörigen **Ground-Truth**.

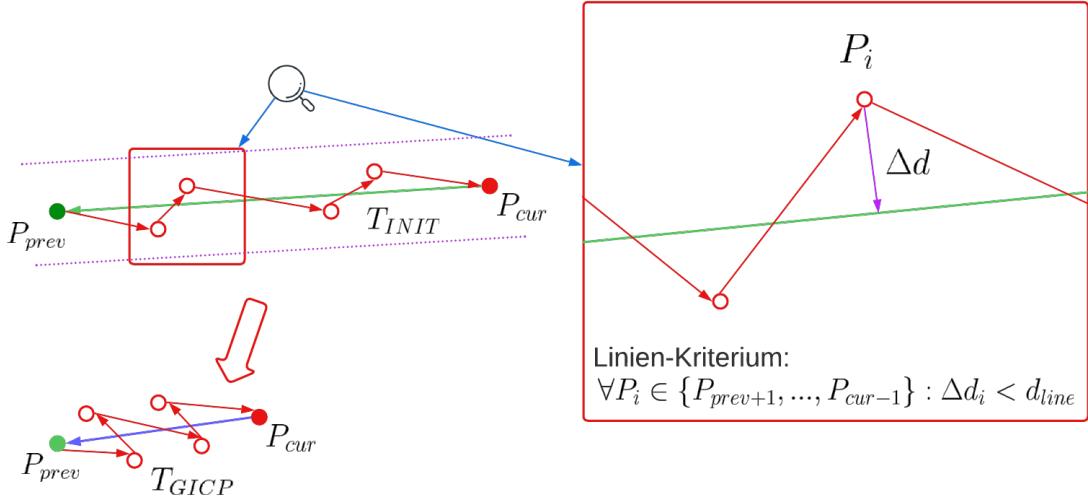
mit wenigen Features, fehlerhaft. Hier sind zwar die genannten Rahmenbedingungen erfüllt, allerdings verschlechtern die Schleifenschlüsse das Ergebnis maßgeblich. Um diese Problem zu lösen wurden mehrere Ansätze implementiert und evaluiert. Dazu wurde zunächst das Problem des Scan-Matching näher betrachtet. Dabei stellt sich heraus, dass für eine gute Transformations-Schätzung der Scan-Matching Algorithmen neben einer guten Initial-Schätzung zusätzlich eine dichtere Model-Punktwolke essentiell ist. Basierend auf der Annahme, dass eine lokale Umgebung  $\{P_{i-x}, \dots, P_i, \dots, P_{i+x}\}$  um eine Pose  $P_i$  konsistent registriert ist, wurde die Model-Punktwolke um die Punktwolken aus der direkten Nachbarschaft, gegeben durch den Parameter  $x$ , angereichert. Dazu wird jeweils die Pose-Differenz zur Model-Punktwolke berechnet und die Punktwolke entsprechend der berechneten Transformation ins Koordinatensystem der Model-Punktwolke angereichert. Ergebnis ist eine deutlich dichtere Punktwolke, was die Korrespondenzfindung für die Scan-Matching Algorithmen deutlich verbessert und so zu einem besseren Endergebnis führt. Abbildung 5.2 zeigt die durchschnittlichen Fitness-Scores aller Schleifenschluss-Kandidaten der jeweiligen Iteration für ein durch  $x$  gegebenes Fenster um die Model-Punktwolke herum.

Ein weiteres, bereits angeschnittenes Problem sind Schleifenschlüsse, die zwar die vorgegebenen Rahmenbedingungen erfüllen, allerdings grundsätzlich fehlerhaft sind. Hier ist zum Beispiel in einem Flur das Scan-Matching aufgrund einer sehr Feature armen Umgebung konvergiert und die Validierung auf Basis des Fitness-Scores konnte zusätzlich keinen Fehler bei der Registrierung identifizieren. Dann werden häufig Positionen, die in einer initialen Schätzung weit auseinander liegen aufeinander gezogen. Zusätzlich kann es vorkommen, dass das Scan-Matching mit einem guten Fitness-Score konvergiert, obwohl die Umgebungen nicht zueinander passen. Die berechnete finale Transformation ist in diesem Fall so fehlerhaft, dass sie nicht mit der initialen Schätzung vereinbar ist. Aus diesem Grund wurden zwei Validatoren beziehungsweise **Rejector** entwickelt, die neben dem Fitness-Score zusätzliche Validations-Stufen einführen. Dies ist zum einen der **Linien-Rejector** und auf der anderen Seite der **Reichweiten-Rejector**. Diese nutzen die Unsicherheiten der einzelnen Posen aus. Vor Beginn des Algorithmus erfolgt eine Einschätzung über die Sicherheit der initial bestimmten Posen. Diese kann beliebig schlecht gewählt werden um sicherzustellen, dass bei der Optimierung gewünschte Änderungen vorgenommen werden. Dies erhöht allerdings die Wahrscheinlichkeit für fehlerhafte Schleifenschlüsse, die durch die beiden im Folgenden beschriebenen Validatoren nicht erfasst werden können. Kann für eine Initial-Schätzung zum Beispiel aufgrund voriger Erfahrungen ein gewisser Fehler für die Translation und Rotation zwischen aufeinander folgenden Posen abgeschätzt werden, sollte dieser im Folgenden verwendet werden.

### **Linien-Rejector**

Je nach Parametrisierung ist es möglich, dass Schleifenschlüsse zwischen Posen  $P_{cur}$  und  $P_{prev}$  identifiziert werden, deren Zwischen-Posen näherungsweise auf einer Linie liegen. Der maximale Abstand einer Zwischen-Pose zur durch  $P_{cur}$  und  $P_{prev}$  definierten Linie ist mit  $d_{line}$  definiert und parametrisierbar. Standardmäßig ist  $d_{line} = 0.5m$ . Eine beschriebene Identifikation ist im Regelfall möglich, wenn, wenn  $d_{max} > d_{trav}$ . In Ausnahmefällen kann dies bei bestimmten Trajektorien schon für geringere  $d_{trav}$  der Fall sein. Diese Art von Schleifenschlüssen stellt im Grundsatz kein Problem dar, allerdings ist es durch die Bewertung des Ergebnisses des Scan-Matching anhand des Fitness-Scores besonders in Fluren oder ähnlichen Regionen, die unvorteilhaft für ein Scan-Matching sind, möglich, dass am Schleifenschluss beteiligte Posen ohne Rücksicht auf die initialen Schätzungen und Zwischen-Posen aufeinander gezogen werden. Abbildung ?? zeigt dieses dies schematisch und nimmt zusätzlich Bezug auf die Identifikation von diesen besonderen Fällen.

Wie bereits zuvor beschrieben, ist eine mögliche Lösung dieses Problems die Ausnutzung von relativen, lokalen Unsicherheiten zwischen aufeinanderfolgenden Posen. Der Linien-Rejector nutzt dabei die definierten Unsicherheiten für die Translation als Vektor aus den gegebenen Standardabweichungen:  $\vec{\sigma}_t = (\sigma_x, \sigma_y, \sigma_z)$ . Diese können entweder konstant gegeben oder für jede Pose-Differenz aufeinander folgender Posen einzeln, zum Beispiel anhand von Berechnungen im SLAM Ansatz, gegeben sein. Im Folgenden werden die Standardabweichungen als konstant angenommen. Gegeben  $P_{cur}$  und  $P_{prev}$ , die Anzahl an Zwischen-Posen  $N_{between}$ , sowie die konstante Standardabweichung der Translation  $\vec{\sigma}_t$  lässt sich nun Konfidenzintervall berechnen, in dem basierend auf der gegebenen Standardabweichung ein gewisser Prozentsatz der für  $P_{cur}$  möglichen liegt. Dies ist in 3D ein räumlicher Bereich. Als Erwartungswert wird die initiale



**Abbildung 5.3:** Diese Grafik beschreibt die Funktionsweise des Linien-Rejectors. Zusätzlich definiert sie das mathematische Kriterium für eine approximative Linie, gegeben  $d_{line}$ :  $\forall P_i \in \{P_{prev+1}, \dots, P_{cur-1}\} : \Delta d_i < d_{line}$ . Die Abbildung ist zweigeteilt. In der rechten Vergrößerung eines Teilbereichs der oben links dargestellten Trajektorie ist das Linien-Kriterium dargestellt. Die Linie zwischen den beiden Posen  $P_{cur}$  und  $P_{prev}$  ist oben links und in der Vergrößerung in grün dargestellt. Sie wird durch den Translationsanteil von  $T_{INIT}$ , der initial gegebenen Transformation zwischen  $P_{cur}$  und  $P_{prev}$ , sowie den Translationsanteil von  $P_{cur}$  als Stützvektor gegeben. Durch lila gestrichelte Linien ist die Umgebung dargestellt, in der sich das Sensorsystem befindet. In diesem Fall handelt es sich um einen besonders Feature-armen Flur. Bei der Validierung mittels Scan-Matching (in diesem Fall per GICP) wurde die durch  $T_{GICP}$  beschriebene Transformation ermittelt. In diesem Fall ist das Scan-Matching mit einem sehr guten Fitness-Score konvergiert, obwohl die Transformation augenscheinlich fehlerhaft ist. Dies liegt an der beschriebenen Umgebung, die aus Sicht des Laserscanners für verschiedene Posen sehr ähnlich, wenn nicht nahezu identisch aussieht. Dies sind sehr schlechte Voraussetzungen für ein Scan-Matching. Unten links ist die zu erwartende Optimierung der Trajektorie auf Basis der fehlerhaften Transformation  $T_{GICP}$ . Bei den Zwischen-Posen tritt durch die Stauchung der Geraden ein Ziehharmoneika-Effekt auf. Dies hat fatale Auswirkungen auf die generierte Karte und den weiteren Verlauf der Trajektorie.

Schätzung für  $P_{cur}$ , genauer gesagt dessen Translationsanteil  $\vec{t}_{cur}$  verwendet. Damit ergibt sich für die untere Grenze  $\vec{t}_u$  und obere Grenze  $\vec{t}_o$  des Konfidenzintervalls unter Ausnutzung der akkumulierten Standardabweichungen zwischen  $P_{prev}$  und  $P_{cur}$ :

$$\vec{t}_u = \vec{t}_{cur} - (N_{between} + 1) \cdot k \cdot \vec{\sigma}_t \quad (5.4)$$

$$\vec{t}_o = \vec{t}_{cur} + (N_{between} + 1) \cdot k \cdot \vec{\sigma}_t \quad (5.5)$$

Dabei bezeichnet  $k$  die verwendete standardisierte Intervallgrenze. Im Folgenden wird hierfür 1 verwendet, eine beliebige Anpassung dieses Wertes ist möglich. Für  $k = 1$  werden für eine gegebene Standardabweichung im Konfidenzintervall etwa 80% bis 85% der möglichen relativen

Pose-Änderungen zwischen aufeinander folgenden Posen abgedeckt. Der Prozentsatz des durch  $\vec{t}_u$  und  $\vec{t}_o$  abgedeckten Volumens berechnet sich aus folgender Potenz:

$$\prod_{i=0}^{N_{between}} 0.8 \quad (5.6)$$

Zugrunde liegt die Schätzung des Prozentsatzes, der durch die Intervallgrenzen, gegeben  $k$  abgedeckt wird (hier etwa 80%). Für 4 Zwischen-Posen beträgt die Abdeckung in diesem Fall in etwa 33%. Für eine größere Abdeckung ist ein größeres  $k$  zu wählen. Mit zunehmender Anzahl an Zwischen-Posen wird der durch den Rejector abgedeckte Bereich verhältnismäßig immer kleiner, obwohl er linear vergrößert wird. Dies hat eine sehr strenge Behandlung von Schleifenschlüssen auf Linien zur Folge. Mit einem größeren  $k$  kann dies entschärft werden. Liegt  $P_{cur}$  nach Bestimmung der neuen Transformation  $T_{GICP}$  außerhalb des durch  $\vec{t}_u$  und  $\vec{t}_o$  beschriebenen Volumens, der gefundene Schleifenschluss als ungültig angesehen. Liegt die neue Pose innerhalb des Volumens, ist sie validiert. Auf diese Weise konnten in einem Beispieldurchlauf für eine Fenster-Größe von  $x = 0$  und  $\Delta d = 2.0m$  bei einem Datensatz mit 200 Posen 8 fehlerhafte Schleifenschlüsse identifiziert und verworfen werden. Mit größerer Fenstergröße reduziert sich die Anzahl dieser fehlerhaften Schleifenschlüsse. So konnten bereits bei einer Größe von  $x = 2$  in diesem Datensatz keine fehlerhaften Schleifenschlüsse identifiziert werden. Diese sind jedoch auch weiterhin nicht ausgeschlossen, weshalb der Rejector als weitere Sicherheitsschranke hinter das Scan-Matching geschaltet wird.

### **Reichweiten-Rejector**

Der **Reichweiten-Rejector** basiert auf dem identischen Prinzip wie der zuvor erläuterte Linien-Rejector. Hier wird zusätzlich zur Standardabweichung in der Translation auch die Standardabweichung in der Rotation berücksichtigt. Zusätzlich ist das Vorhandensein einer Linie keine Voraussetzung für diesen Rejector. Alle anderen mathematischen Verhältnisse aus dem oben vorgestellten Rejector treffen so auch hier zu und werden analog sowohl für die Translation, als auch für die Rotation angewandt. Dieser Rejector greift ist dem Linien-Rejector nachgeschaltet.

Der nachfolgende Abschnitt beschäftigt sich mit der Optimierung des Pose-Graphen auf Basis identifizierter Schleifenschlüsse.

## **5.2 Optimierung des Pose-Graphen**

Für die Optimierung des Pose-Graphen wird in diesem Fall die C++ Bibliothek GTSAM [9] verwendet. GTSAM implementiert die Fusion von Sensor Daten für Anwendungen im Bereich von Robotik und Computer-Grafik und kommt insbesondere in SLAM-Ansätzen zum Einsatz [9]. Als unterliegende Datenstruktur für den hier verwendeten Anwendungsfall dient ein Faktorgraph. Faktorgraphen sind grafische Modelle, die sich gut für komplexe Schätzungsprobleme wie SLAM und deren Modellieren eignen. Ein Faktorgraph ist ein bipartiter beziehungsweise paarer Graph, der Beziehungen zwischen zwei Mengen darstellt. Der Graph besteht aus Faktoren beziehungsweise **Graph-Constraints**, die mit der Menge der **Variablen** verbunden sind. Dabei stellen die

Variablen die unbekannten Zufallsvariablen des Schätzproblems dar, während die Faktoren probabilistische Beschränkungen oder Voraussetzungen für diese Variablen darstellen, die aus Messungen wie Odometrie, **IMU (Intertial Measurement Unit)**-Schätzungen, gesichteten Landmarken oder SLAM-Schätzungen abgeleitet werden [9]. Im Folgenden sind die Zufallsvariablen des Schätzproblems die gesuchten optimierten Posen des Pose-Graphs. Als Faktoren werden die initialen Pose-Schätzungen verwendet, die durch eine zusätzliche Vorregistrierung mittels GICP verbessert werden, sowie die identifizierten Schleifenschlüsse. GTSAM stellt für diese Faktoren vordefinierte Klassen, die in Tabelle 5.1 vorgestellt werden.

In den Faktor-Graph selbst werden beim Durchlauf des Algorithmus lediglich die beschriebenen Faktoren eingefügt. Zu einem beliebigen Zeitpunkt besteht er aus dem initialen Prior-Faktor der ersten Pose, sowie mehreren Between-Faktoren für aufeinander folgenden Posen und Schleifenschlüsse. Der Verbund mit der Menge der Variablen wird durch die Verwendung von Indices erreicht. Dabei erhält der Prior-Faktor den Index 0, der er sich lediglich auf die erste Pose des Pose-Graphen bezieht, die mit 0 indiziert ist. Between-Faktoren erhalten jeweils zwei Indices, die ebenfalls den entsprechenden Posen des Pose-Graphen zugeordnet werden können. Die beiden Indices legen fest, zwischen welchen Variablen der Faktor gilt und in welche Richtung er gilt. Der grundlegende Ablauf bei dem Einfügen neuer Faktoren in den Faktorgraphen und der anschließenden Optimierung der Variablenmenge, hier dem verwendeten Pose-Graphen, ist in Abbildung 5.4 dargestellt.

Im Folgenden wird der hier vorgestellte Ansatz auf Basis mehrerer Datensätze evaluiert.

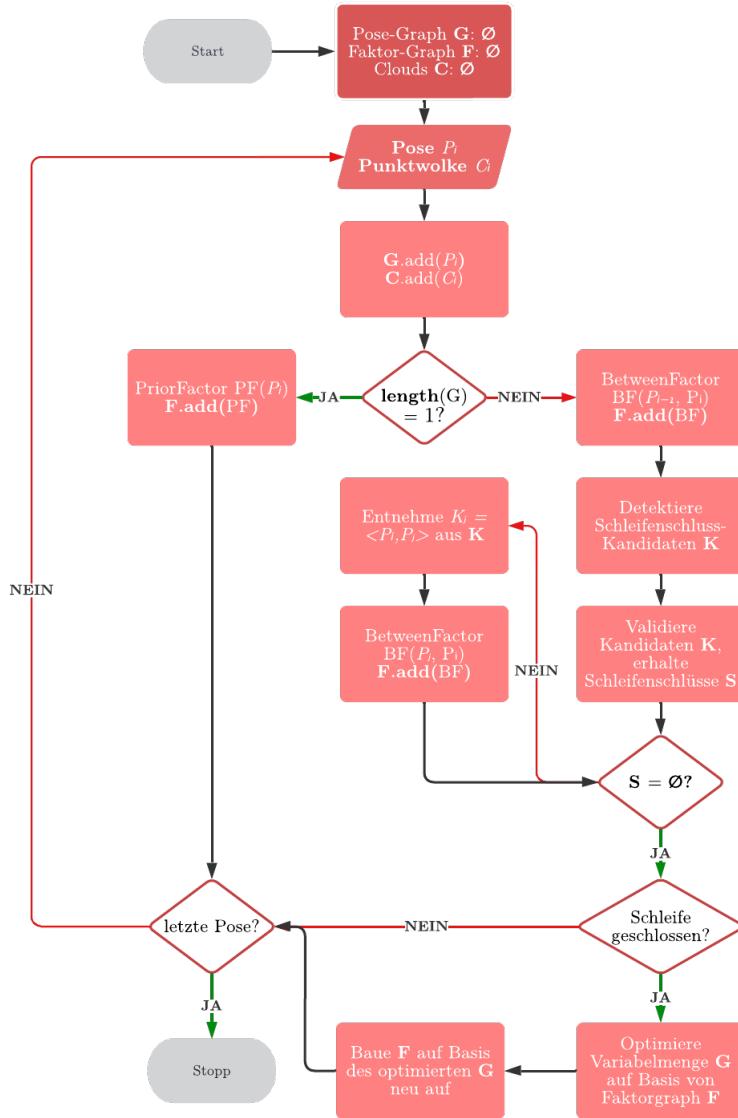
### 5.3 Evaluation der Graph-Optimierung

Dieser Absatz befasst sich mit der Evaluation des zuvor vorgestellten Ansatzes zur Optimierung des Pose-Graphen mittels identifizierter Schleifenschlüsse unter Verwendung der GTSAM Bibliothek [9]. Wie beschrieben werden zur Evaluation des Ansatzes mehrere Datensätze verwendet, die aus einer Initialschätzung des Pose-Graphen mit einer zu jeder Pose  $P_i$  zugehörigen Punktwolke  $C_i$ , die relativ zum Koordinatensystem der zugehörigen Pose gesehen ist. Zu den verwendeten Datensets gehört sowohl der **Hannover-1** Datensatz der Universität Osnabrück mit zugehöriger Initialschätzung des Pose-Graphen durch Odometrie und einer **Ground-Truth**, die in [24] vorgestellt wird. Die Ground-Truth bezeichnet eine Pose-Historie, die die (näherungsweise) realen Posen enthält, von denen die Daten aufgenommen wurden. Dies erlaubt einen direkten Vergleich der mit diesem Ansatz optimierten Trajektorie mit der Ground-Truth. Zusätzlich wurden im Laufe dieser Arbeit weitere Datensätze aufgenommen. Aus diesen Datensätzen erfolgte zunächst die Bestimmung einer Initialschätzung des Pose-Graphen mit einer überarbeiteten Variante des in [27] vorgestellten SLAM-Ansatzes. Bei einer visuellen Überprüfung der so generierten Initialschätzungen fallen direkt mehrere Fehler wie doppelte Wände auf, die durch eine Fehlerfortpflanzung beim SLAM auftreten. Im Zuge dieser Evaluation soll überprüft werden ob diese Fehler durch diesen Ansatz kompensiert werden können.

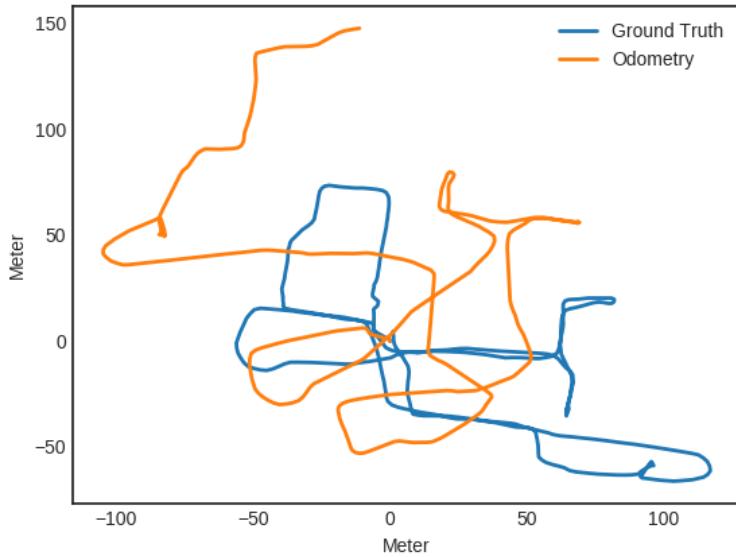
#### 5.3.1 Datensatz: Hannover-1

Zunächst erfolgt eine Evaluation des Hannover-1 Datensatzes. Abbildung 5.5 zeigt eine Ge-

eingehen auf  
inkompatible  
Koordinatensystem  
Situations und  
entsprechend  
notwendige  
Transformation



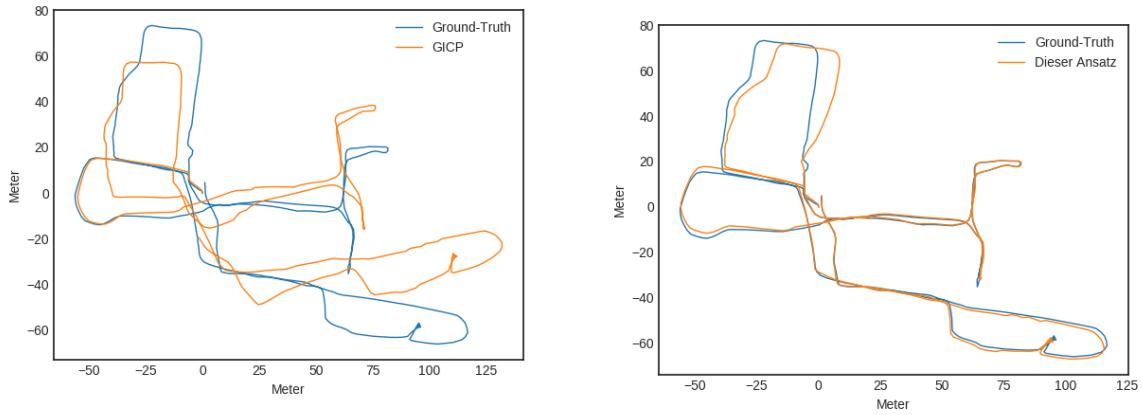
**Abbildung 5.4:** Dieses Flussdiagramm beschreibt den algorithmischen Ablauf der Graph-Optimierung unter besonderer Betrachtung der in den Faktorgraphen einzufügenden Faktoren. Für den Fall, dass Schleifenschlüsse gefunden werden, wird die Menge der Variablen, hier der Pose-Graph auf Basis der in den Faktorgraphen eingefügten Faktoren optimiert. Als Optimierer wird hier der in GTSAM implementierte **Levenberg-Marquardt-Optimizer** verwendet, der die von Levenberg [14] und Marquard [17] vorgeschlagene Lösung von nicht-linearen **Least-Squares** Problemen implementiert. Im Anschluss an die durchgeführte Optimierung, muss der Faktor-Graph  $F$  auf Basis der neuen Pose-Differenzen in  $G$  neu aufgebaut werden. Dies wird so lange durchgeführt, bis das letzte Pose-Punktwolke Paar der initialen Schätzung abgearbeitet ist. Um die Pose-Differenzen der initialen Schätzung auch nach der Optimierung des Teilgraphen beim Einfügen einer neuen Pose  $P_i$  zu erhalten wird mit **Pose-Deltas** gearbeitet. Dazu wird immer die zuletzt eingefügte Pose der initialen Schätzung gespeichert, bei einer neuen Pose das Pose-Delta beziehungsweise die Pose-Differenz zwischen der neuen und der gespeicherten Pose berechnet und auf die zuletzt eingefügte Pose in  $G$  angewandt.



**Abbildung 5.5:** Diese Grafik zeigt die Ground-Truth des Hannover-1 Datensatzes im Vergleich mit der initialen Odometrieschätzung. Es ist ersichtlich, dass die Odometrieschätzung signifikante Fehler aufweist, sodass die Trajektorien stark divergieren.

genüberstellung der beschriebenen Ground-Truth des Datensatzes mit der durch Odometrie gegebenen Initialschätzung. Hier ist eine deutliche Differenz zwischen den beiden Pose-Graphen zu erkennen, die nicht ausschließlich durch die Integration von Schleifenschlüssen aufgelöst werden kann. Aus diesem Grund wird eine Vorregistrierung der aktuell betrachteten Initialschätzung der Pose  $P_i$  mit den zuvor in den Pose-Graphen eingefügten Posen mittels GICP vorgenommen. Lediglich für die erste Pose wird die initiale Schätzung verwendet. Wie zuvor erläutert und in Abbildung 5.2 schematisch dargestellt ist, verbessert eine Anreicherung der Model-Punktwolke mit den Punktwolken der benachbarten Posen das Ergebnis der Registrierung maßgeblich. Diese Erkenntnis kommt auch bei der Vorregistrierung zum Einsatz. Der erörterte Vorregistrierungsschritt findet zusätzlich bei den anderen Datensätzen Verwendung, um eine lokale Konsistenz der Daten zu gewährleisten.

Um die Effektivität der Schleifenschlüsse im hier vorgestellten Ansatz herauszustellen, erfolgt ein Vergleich mit einer Registrierung des Datensatzes durch GICP. Dies ist in Abbildung 5.6 dargestellt. Zu erkennen ist eine deutliche Optimierung des Pose-Graphen durch die Einführung der Schleifenschlüsse. Lediglich eine Teilschleife des Graphen weicht im Vergleich stärker von der Ground-Truth ab. Hier lag zunächst ein großer Fehler in der Vorregistrierung vor, der von den identifizierten Schleifenschlüssen nur zum Teil korrigiert werden konnte. Dies wird hier zusätzlich anhand einer weiteren Grafik deutlich gemacht, die den absoluten Fehler in der Translation zwischen einer Pose  $P_i$  des mit diesem Ansatz korrigierten Pose-Graphen mit der zugehörigen Pose  $P_i^{GT}$  der Ground-Truth vergleicht. Dies ist in Abbildung 5.7 dargestellt. Dabei ist der



(a) Vergleich einer Registrierung des Datensatzes mittels GICP, basierend auf den initial geschätzten Posen, mit der Ground-Truth.  
 (b) Vergleich des hier vorgestellten Ansatzes mit der Ground-Truth.

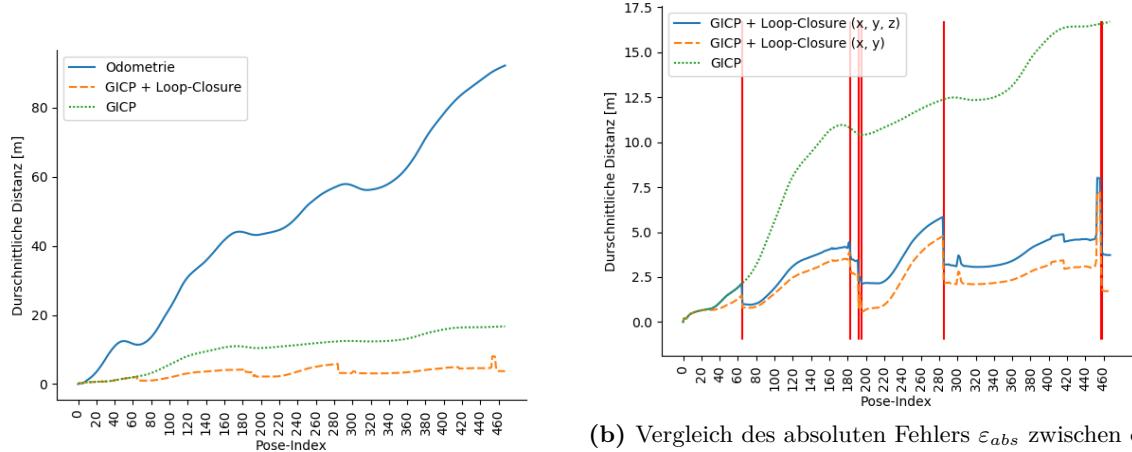
**Abbildung 5.6:** Diese Grafik zeigt einen Vergleich des nur mit GICP registrierten Ergebnisses auf der linken Seite mit dem Ergebnis des hier vorgestellten Ansatzes, der neben einer Vorregistrierung mit GICP zusätzlich Schleifenschlüsse integriert, auf der rechten Seite. Beide Ergebnisse sind jeweils im Vergleich zur genannten Ground-Truth dargestellt. Es ist zu erkennen, dass die Registrierung des Datensatzes mittels GICP, basierend auf den initial geschätzten Posen eine deutliche Verbesserung erzielt, jedoch weiterhin große Abweichung zur Ground-Truth vorliegen. Der hier vorgestellte Ansatz nutzt die durch GICP vorregistrierten Daten und führt zusätzlich eine Optimierung des Graphen mittels Schleifenschlüssen ein. Dies verbessert das Ergebnis unter Betrachtung der Ground-Truth signifikant.

absolute Fehler  $\varepsilon_{abs}$  für die aktuelle Anzahl an Posen  $n_{cur}$ , sowie den Translationsanteilen der Posen des mit diesem Ansatzes korrigierten Pfades  $\vec{t}_i$  und der Ground-Truth  $\vec{t}_i^{GT}$  wie folgt definiert:

$$\varepsilon_{abs} = \frac{\sum_{i=0}^{n_{cur}} \left\| \vec{t}_i^{GT} - \vec{t}_i \right\|}{n_{cur}} \quad (5.7)$$

Die Abbildung 5.7 zeigt dabei die deutlichen Verbesserungen des absoluten Fehlers durch die Nutzung von GICP und die weitere Verbesserung durch die Nutzung von Schleifenschlüssen. Ein direkter Vergleich mit den Ergebnissen aus [24] ist an dieser Stelle nicht möglich, da diese nur für einen anderen Datensatz gegeben sind. In Abbildung A.1 im Anhang ist eine Grafik aus [24] platziert, anhand derer ein visueller Vergleich mit den hier erzielten Ergebnissen ermöglicht wird. Es ist zu erkennen, dass der präsentierte Ansatz in diesem Datensatz an vielen Stellen genauer ist als der Ansatz aus [24]. Daran ändern auch die in [24] genannten Post-Processing Schritte nichts.

Im Folgenden wird der Ansatz auf Basis weiterer Datensätze der Universität Osnabrück validiert.



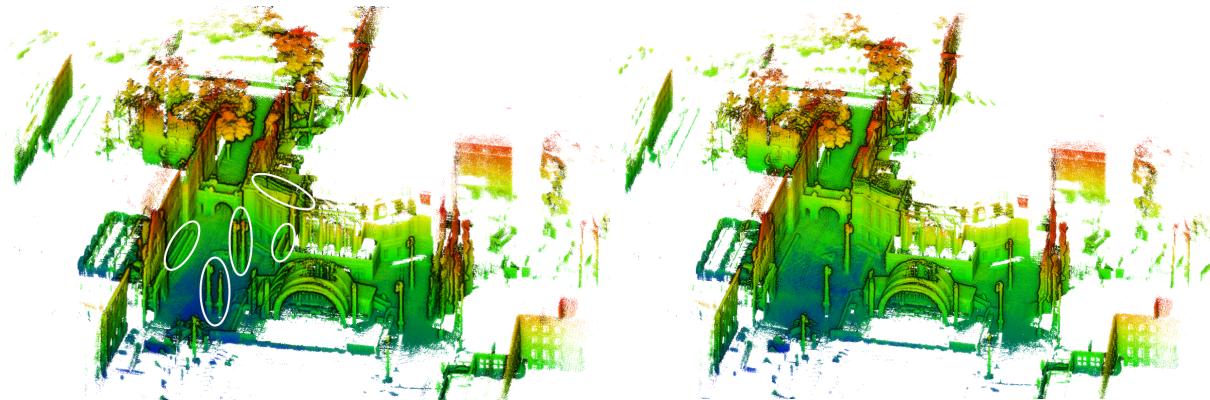
(a) Vergleich des absoluten Fehlers  $\varepsilon_{abs}$  zwischen der initialen Schätzung, des Pose-Graphen nach Registrierung mit GICP und dem Ergebnis des hier vorgestellten Ansatzes mit der Ground-Truth.

(b) Vergleich des absoluten Fehlers  $\varepsilon_{abs}$  zwischen dem Pose-Graphen nach Registrierung mit GICP und dem Ergebnis des hier vorgestellten Ansatzes mit der Ground-Truth. Rote vertikale Linien zeigen dabei die Schleifenschlüsse durch die der absolute Fehler um mindestens 10% reduziert wurde. Die absolute Anzahl an Schleifenschüssen ist deutlich höher.

**Abbildung 5.7:** Diese Grafik vergleicht den absoluten Fehler  $\varepsilon_{abs}$  in der Translation zwischen einer Pose  $P_i$  mit der zugehörigen Pose  $P_i^{GT}$  der Ground-Truth. Zusätzlich zum absoluten Fehler dieses Ansatzes wurde in der rechten Abbildung zusätzlich der absolute Fehler unter Vernachlässigung der z-Achse untersucht. Diese Untersuchung wurde so bereits in [24] durchgeführt, da der aufgenommene Datensatz aufgrund der Konfiguration des Sensorsystems bei der Registrierung aus unbekannten Gründen eine Kurvenform in der z-Achse annimmt. Aus diesen Grund wurde an dieser Stelle ebenfalls eine Projektion der Daten auf die  $z = 0$  Ebene vorgenommen und der entsprechende Fehler verglichen. Dieser ist mit einem durchschnittlichen absoluten Fehler von  $1.7m$  deutlich geringer als der finale Fehler ohne die Projektion, der bei  $3.7m$  liegt.

### 5.3.2 Datensatz: Chemnitzer Oper

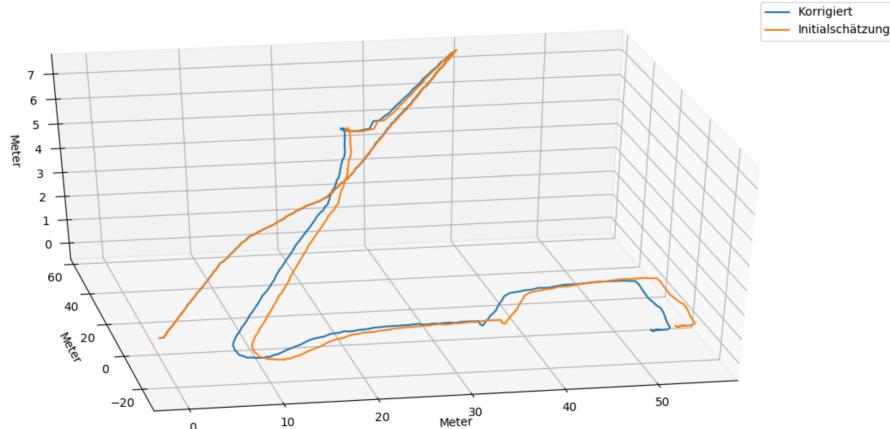
Ein weiterer Datensatz der Universität Osnabrück ist eine Aufnahme des Vorplatzes der Oper in Chemnitz. Grundlage für die Optimierung mit dem hier vorgestellten Ansatz ist eine Initialschätzung des Pose-Graphen mit einer abgewandelten Variante des in [27] vorgestellten SLAM-Ansatzes. Ein Ausschnitt der zur Initialschätzung zugehörigen Punktwolke ist in Abbildung 5.8 dargestellt. Hier wird aufgrund der gekennzeichneten doppelten Wände und Objekte deutlich, dass ein akkumulativer Fehler in der bestimmten Initialschätzung vorliegt. Dieser Fehler soll nun mit Hilfe des hier vorgestellten Ansatzes korrigiert werden. Dazu wird der Pose-Graph wie in Abbildung 5.4 vorgestellt optimiert. Das Ergebnis dieser Optimierung ist die in Abbildung 5.9 gekennzeichnete Anpassung des Pose-Verlaufs. Die Anpassung der Punktwolke auf Basis des korrigierten Pfades wird durch eine Transformation der zu  $P_i$  zugehörigen Punktwolke  $C_i$ . Die dazu verwendete Transformation ist die Transformation von  $P_i$  ins Koordinatensystem der Karte:  $T_{P_i \rightarrow MAP}$ . Das Ergebnis der Korrektur ist ebenfalls in Abbildung 5.8 zu erkennen.



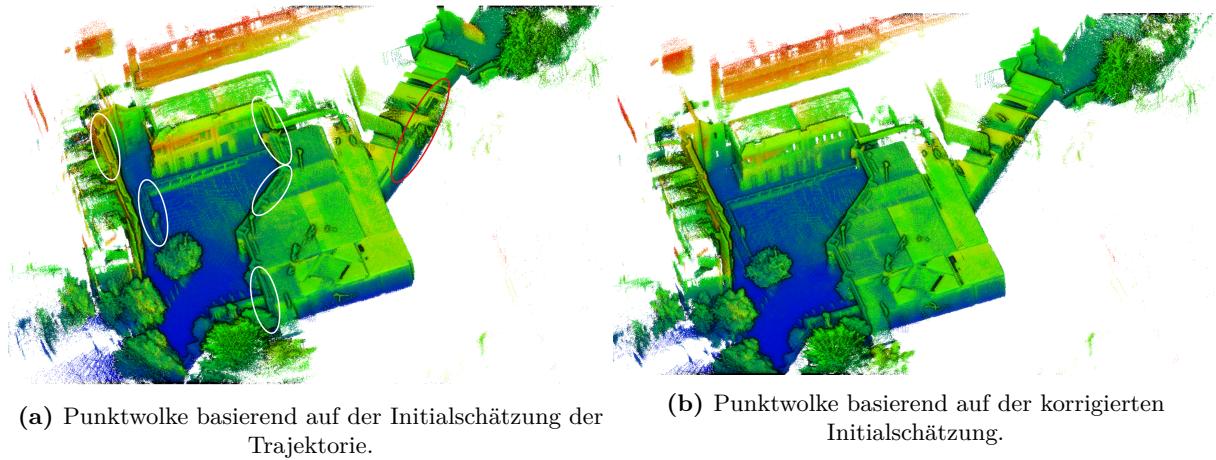
(a) Punktfolke basierend auf der genannten Initialschätzung der Trajektorie.

(b) Punktfolke basierend auf der mit dem hier vorgestellten Ansatz optimierten Trajektorie.

**Abbildung 5.8:** Diese Grafik zeigt einen Vergleich der Punktfolken der Chemnitzer Oper vor und nach der in diesem Ansatz angewandten Korrektur. Zu erkennen ist eine deutliche Verbesserung des Gesamtbildes der Punktfolke durch die Nutzung der Schleifenschlüsse. Doppelte Wände durch akkumulative Fehler konnten nahezu vollständig kompensiert werden.



**Abbildung 5.9:** Grafische Darstellung der Initialschätzung der Trajektorie im Vergleich mit dem korrigierten Pfad. Im linken Teil der Abbildung sind doppelte Objekte und Wände in der Punktfolke gekennzeichnet, die durch einen akkumulativen Fehler im SLAM-Verfahren generiert wurden. Es ist eine deutliche Anpassung der Trajektorie zu beobachten. Diese Anpassung sorgt für die in Abbildung 5.8 erkennbare Kompensation der Fehlerakkumulation und resultiert in einem signifikant besseren Ergebnis, welches keine erkennbaren doppelten Wände ausweist. Die schräge Ausrichtung des Pfades - und damit zusammenhängend auch der Punktfolke - im Raum beruht auf der schrägen Position des Laserscanners. Zum Zeitpunkt der Aufnahme der Daten wurde dieser leicht schräg auf dem Kopf der Person positioniert, die die Daten aufgenommen hat. Dies gilt ebenfalls für den in Abbildung 5.11 dargestellten Verlauf des Pose-Graphen.



**Abbildung 5.10:** Wie Abbildung 5.8 zeigt auch diese Abbildung einen Vergleich der Punktwolken vor und nach Korrektur des in diesem Kapitel vorgestellten Algorithmus. Im linken Teil der Abbildung sind doppelte Objekte und Wände in der Punktwolke gekennzeichnet, die durch einen akkumulativen Fehler im SLAM-Verfahren generiert wurden. Auch hier ist eine signifikante Verbesserung der Trajektorie, erkennbar an der kontinuierlichen Punktwolke, die keine doppelten Wände oder Objekte enthält, zu beobachten.

### 5.3.3 Datensatz: Campus Universität Osnabrück

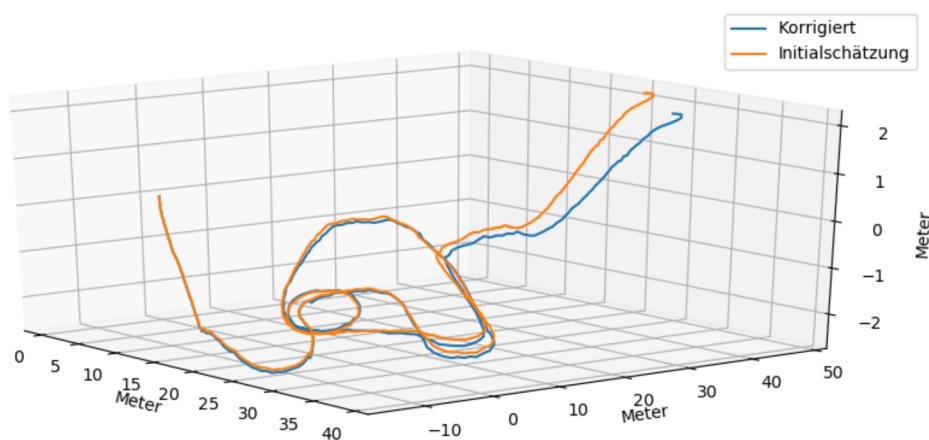
Dieser Absatz behandelt einen weiteren Datensatz unter identischen Bedingungen wie im vorigen Absatz beschrieben. Bei dem Datensatz handelt es sich um einen Teil des Physikgebäudes der Universität Osnabrück inklusive der Tiefgarage mit Durchgang. Die Evaluation dieses Datensatzes erfolgt entsprechend des vorigen Absatzes und ist in Abbildung 5.10 und 5.11 dargestellt. Zu erkennen ist, dass auch hier eine Kompensation der Fehlerfortpflanzung durch den hier vorgestellten Ansatz erreicht werden konnte.

Die in diesem Kapitel entwickelte Lösung wurde hier zunächst vollständig abgekapselt von der TSDF-Karte betrachtet. Dies wird im nachfolgenden Kapitel nun auf Basis der hier vorgestellten Lösung zur Kompensation von akkumulierten Fehler durch die Integration von Schleifenschlüssen und den Ausführungen aus Kapitel 4 weiter verfolgt.

an dieser Stelle könnte noch eine Menge an Abbildungen folgen, die die Ergebnisse in Abhängigkeit von mehreren Ursachen beschreiben

ggf. auf andere Scan-Matching Verfahren eingehen (ICP, GICP, Teaser++, VGICP)

Zeitmessungen?



**Abbildung 5.11:** Grafische Darstellung der Initialschätzung der Trajektorie im Vergleich mit dem korrigierten Pfad. Es ist eine deutliche Anpassung der Trajektorie zu beobachten. Diese Anpassung sorgt für die in Abbildung 5.10 erkennbare Kompensation der Fehlerakkumulation und resultiert in einem deutlich besseren Ergebnis, welches keine erkennbaren doppelten Wände ausweist.

**Tabelle 5.1:** Verwendete GTSAM-Faktoren und deren Funktionsweise.

Faktor	Beschreibung	Nutzung
<b>Prior-Faktor</b>	Definiert ein initiales Rauschen für die erste Pose des Pose-Graphen. Dieser Faktor ist der einzige verwendete Faktor, der einen absoluten Pose-Constraint beziehungsweise Faktor definiert. Alle anderen Faktoren sind relative Faktoren zwischen zwei Posen, gegeben durch ihre Indices. Aus diesem Grund wählt [23] an dieser Stelle ein großes initiales Rauschen, um die Optimierung nicht durch eine zu statische erste Pose einzuschränken. [23] definiert einen 6D Varianz-Vektor, der die Varianzen der Rotations- und Translationskomponenten (in dieser Reihenfolge, in $\text{rad}^2$ und $\text{m}^2$ ) enthält. Dieser lautet wie folgt: $(10^{-2}, 10^{-2}, \pi \cdot \pi, 10^8, 10^8, 10^8)^T$ . Dies wurde in dieser Arbeit zunächst übernommen und später parametrisierbar gemacht. Die in dieser Arbeit verwendeten Default-Werte lauten: $(10^{-2}, 10^{-2}, 10^{-2}, 0.064, 0.064, 0.064)^T$ . Diese orientieren sich an dem Ansatz von [10].	Der Prior-Factor wird insgesamt einmal für das definierte initiale Rauschen der ersten Pose verwendet.
<b>Between-Faktor</b>	Definiert einen relativen Pose-Constraint zwischen zwei Posen $P_i$ und $P_j$ , gegeben durch deren Indices $i$ und $j$ . Dieser Constraint ist gegeben als 3D Pose-Differenz, bestehend aus Rotation und Translation. Hierfür stellt GTSAM die Klasse <i>Pose3</i> zur Verfügung. Auch für diesen Faktor kann neben dem relativen Constraint zwischen zwei Posen auch eine Unsicherheit definiert werden. Für aufeinanderfolgende Posen ist diese parametrisch gegeben (Default: $(10^{-2}, 10^{-2}, 10^{-2}, 0.064, 0.064, 0.064)^T$ ), für identifizierte Schleifenschlüsse wird der beim Scan-Matching resultierende Fitness-Score $\Gamma$ verwendet: $(\Gamma, \Gamma, \Gamma, \Gamma, \Gamma, \Gamma)^T$ . Bei dieser Entscheidung wurde sich an [23] orientiert.	Dieser Faktor findet sowohl zur Beschreibung der Pose-Differenzen aufeinanderfolgender Posen (gegeben durch die gegebenenfalls mit GICP optimierte, initiale Schätzung), als auch für die Beschreibung der im Scan-Matching ermittelten Pose-Differenz zwischen den an Schleifenschlüssen beteiligten Posen.

## Kapitel 6

# Karten-Update Strategien

Dieses Kapitel basiert auf den Ausführungen in Kapitel 4. Dabei wurde herausgestellt, dass ein Update der Karte ohne eine zusätzliche Datenbasis in der Nachbehandlung aufgrund einer Vielzahl an Problemen nicht möglich ist. Im Folgenden wird das Problem auf der um die Menge an Punkt-  
wolken  $\mathbb{C}$  erweiterten Datenbasis neu betrachtet. Dabei wird das Problem selbst neu aufgerollt.  
Anstelle in einem fertigen Pose-Graphen wie in Kapitel 5 beschrieben nach Schleifenschlüssen zu suchen, den Graphen auf Basis der identifizierten Schleifenschlüsse zu optimieren und darauf beruhend ein Update auf einer bereits fertigen Karte durchzuführend wird nun ein inkrementelles Vorgehen angestrebt. Ziel ist die Möglichkeit zu bieten, den Ansatz in einen existierenden, auf einer TSDF-Karte beruhenden, SLAM-Algorithmus zu integrieren. In einer Nachbearbeitung ist es nicht zwingend notwendig die Karte nach jedem identifizierten Schleifenschluss und der darauf beruhenden Graph-Optimierung ein Update der Karte durchzuführend, da die TSDF-Karte selbst für den vorgestellten Optimierungsprozess, der auf den Punkt-  
wolken  $\mathbb{C}$  beruht, nicht benötigt wird. Innerhalb eines auf der TSDF-Karte operierenden SLAM-Ansatzes wie [10] ist es allerdings unerlässlich, dass die Karte in jedem Inkrement auf dem neusten Stand ist um eine konsistente Registrierung neuer Daten an die Karte gewährleisten zu können.

Um dieses Ziel zu erreichen, muss nach jedem Inkrement, in dem eine Graph-Optimierung vorgenommen wurde ein Update der bisher erstellten Karte durchgeführt werden. Da eine sichere und lückenlose Ermittlung von Datenassoziationen basierend auf den Posen  $\mathbb{P}$  und der TSDF-Karte nicht ohne weiteres möglich ist, ist an dieser Stelle eine andere Herangehensweise an das Problem des Updates gefragt. In einem ersten Schritt wird hier ein **Greedy (gierig)**-Ansatz gewählt. Ziel eines solchen Ansatzes ist ein Problem auf eine möglichst einfache Weise zu lösen und dabei längere Laufzeiten der gefundenen Lösung in Kauf zu nehmen. Eine möglichst einfache Lösung beim Update der Karte besteht darin zunächst vollständig auf lokale Updates zu verzichten und stattdessen die gesamte Karte zu löschen und neu zu generieren. Dies ist bezogen auf die Laufzeit besonders bei großen Datensätzen keine gute Idee, da auch Teile der Karte vollständig gelöscht und neu generiert werden, die keines Updates bedürfen. Aus diesem Grund wird im weiteren Verlauf dieses Kapitel zusätzlich auf ein partielles, lokales Update der Karte eingegangen und die benötigten Voraussetzungen und Fallstricke beschrieben. Ziel ist hier eine erhebliche

Reduktion der Laufzeit. Zunächst wird im folgenden Abschnitt auf das beschriebene globale Update eingegangen.

## 6.1 Globales Update der Karte

Dieser Abschnitt befasst sich mit dem beschriebenen Greedy-Ansatz zur Lösung des Kartenupdates. Ziel ist das Löschen der gesamten TSDF-Karte und die nachfolgende Regeneration der Karte basierend auf der erweiterten Datenbasis. Im ersten Schritt wird betrachtet, wie die Karte am effizientesten gelöscht werden kann. Hier lohnt sich eine Betrachtung des Datenflusses zwischen der lokalen und globalen Karte. Dieser wurde in Kapitel ?? bereits kurz angeschnitten. Die globale Karte kapselt die Funktionalität zur Serialisierung und Deserialisierung von Daten basierend auf der HDF5-Datei als Speichermedium. Die globale Karte tauscht die benötigten Daten Chunk weise mit der HDF5 aus. Jeder Chunk beinhaltet eine fest definierte Anzahl an TSDF-Gewichten und zugehörigen Werten und deckt einen vorgegebenen Bereich der globalen Karte ab, der über den Index des Chunks definiert ist. Die lokale Karte ist ein 3D Fenster mit vorgegebenen Seitenlängen, welches den aktuell betrachteten Teil der globalen Karte enthält und über die globale Karte wandern kann. Die lokale Karte liefert dabei eine Abstraktion zur globalen Karte. Grundsätzlich werden neue Daten nur über die lokale Karte geschrieben und schlussendlich an die globale Karte zurückgeliefert, die diese gegebenenfalls serialisiert. Dieses Konstrukt sorgt für eine geringe Auslastung des Arbeitsspeichers, da immer nur ein Teil der gesamten Karte aus der HDF5 geladen wird. Die Abstraktion über die lokale ist sinnvoll, da in der Regel nur einzelne TSDF-Zellen beschrieben werden. An dieser Stelle ist es allerdings von Nöten, dass alle Chunks der globalen Karte gelöscht werden, beziehungsweise die enthaltenen Daten mit Default-Werten ersetzt werden. HDF5 erlaubt zwar ein Löschen einzelner Datensätze (in diesem Fall Chunks), dies gibt allerdings nicht den von dem Datensatz beanspruchten Speicher wieder frei. Dies sorgt dafür, dass die Karte bei einer Löschung aller Datensätze und der darauf folgenden Regeneration der Karte und Chunks auf Basis des neuen Pose-Graphen stetig wächst. Das Beschreiben aller TSDF-Zellen über die Abstraktion der lokalen Karte ist zusätzlich keine gute Idee, da dies hohe Laufzeiten durch eine große Anzahl von Schreibbefehlen und mehrere Verschiebungen (**Shifts**) der lokalen Karte zur Folge hat. Aus diesem Grund wird an dieser Stelle auf die Abstraktion der lokalen Karte verzichtet und innerhalb der globalen Karte eine Methode entwickelt, die alle Chunks auf ihren Ursprungszustand zurücksetzt und mit Default-Werten füllt. Diese Methode ist um ein Vielfaches schneller als das Zurücksetzen der Karte über die Abstraktion der lokalen Karte und die damit verbundenen, benötigten Shifts der lokalen Karte. Doch auch diese Methode sorgt gegebenenfalls für leere Chunks, die nie zur Verwendung kommen, aber Speicher einnehmen. Eine drastische, aber auch schnellere Lösung ist das Löschen der gesamten HDF5 und der anschließenden Neuerstellung der Datei unter demselben Namen mit identischen Metadaten.

Im Anschluss an die Neuerstellung der HDF5 gilt es nun, die Karte auf Basis des neuen Pfades  $\mathfrak{P}_{new}$  zu regenerieren. Dazu ist keine Berechnung der Transformationen zwischen altem und neuen Pfad oder eine Transformation der zu den Posen zugehörigen Punktwolken notwendig, da die Punktwolken jeweils relativ zu den Posen gesehen werden und entsprechend relative Punktkoordinaten enthalten. Es ist für jede Pose, beginnend bei der ersten Pose des neuen

gfg. erklären,  
dass auch das  
wegen HDF5  
nicht geht, weil  
die Dateien  
nicht geschlos-  
sen werden  
und auch  
manuell nicht  
geschlossen  
werden können

Pfades, ein Shift der lokalen Karte zur aktuellen Pose und darauf folgend ein TSDF-Update basierend auf der zugehörigen Punktwolke durchzuführen. Für genanntes TSDF-Update wird hier wieder die Abstraktion über die lokale Karte verwendet. Dies ist zur Veranschaulichung als Pseudo-Code in Abbildung 3 dargestellt.

---

**Algorithm 3** Globales TSDF-Kartenupdate

---

```

1: procedure GLOBALUPDATE(  $map_{global}, map_{local}, \mathfrak{P}_{new}, \mathbb{C}$  )
2:   Lösche die zu  $map_{global}$  zugehörige HDF5
3:   Regeneriere besagte HDF5 mit identischen Metadaten
4:   for  $P_i$  in  $\mathfrak{P}_{new}$  do
5:     Verschiebe  $map_{local}$  nach  $P_i$ 
6:     Führe ein TSDF-Update basierend auf  $P_i$  und der zugehörigen Punktwolke  $C_i$  aus
7:     Schreibe die Daten aus  $map_{local}$  zurück in  $map_{global}$ 
8:   end for
9: end procedure

```

---

Dieses Update sorgt für eine konsistente Karte entsprechend des neuen Pfades, ist aber aufgrund des Löschens und Regenerierens von Daten die keiner Regeneration bedürften sehr Laufzeit intensiv. Dies ist besonders bei großen Umgebungen mit vielen Posen merklich. Besonders vor dem Hintergrund der für einen Shift der lokalen Karte benötigten Laufzeit von häufig mehr als einer Sekunde, abhängig von der Kartengröße der Größe des betroffenen Bereichs. Um diese hohe Laufzeit zu reduzieren und nur den Teil der Karte zu Updaten, der von den Optimierungen des Pose-Graphen betroffen ist, wird im Folgenden auf ein partielles Update eingegangen. Dabei wird der zuvor vorgestellte Ansatz als Base-Line für die Evaluation verwendet, da keine anderen vergleichbaren Ansätze existieren und der beschriebene Ansatz die einfachste Art ist das angestrebte Update der Karte mit den gegebenen Mitteln durchzuführen.

## 6.2 Partielles Update der Karte

Dieser Absatz befasst sich mit einem partiellen Karten-Update der durch Optimierungen am Pose-Graphen betroffenen Teilbereiche der TSDF-Karte. Dazu gilt es zunächst die Teile der Karte zu bestimmten, die ein genanntes Update benötigen. Diese Teile der Karte ergeben sich aus den verschobenen Posen des Pose-Graphen. Jedoch sollte nicht bei jeder Transformation einer Pose ein Update des entsprechenden Kartenbereichs durchgeführt werden. Dazu zählen zum Beispiel Transformationen die nahezu keine Verschiebung der entsprechenden Pose bewirken. Dazu werden an dieser Stelle zwei Schwellen für die minimale Translation  $t_{min}$  und die minimale Rotation um eine beliebige Achse  $rot_{min}$  definiert. Im Anschluss wird für jedes Pose-Paar aus alter Pose  $P_i^{old}$  und neuer Pose  $P_i^{new}$  nach der Graphen-Optimierung die relative Transformation  $T_{P_i^{old} \rightarrow P_i^{new}}$  von  $P_i^{old}$  nach  $P_i^{new}$  ermittelt. Diese ergibt sich aus:

$$T_{P_i^{old} \rightarrow P_i^{new}} = \left( T_{P_i^{new} \rightarrow MAP} \right)^{-1} \cdot T_{P_i^{old} \rightarrow MAP} \quad (6.1)$$

Für jede ermittelte relative Transformation wird nun der Translations- und Rotationsanteil mit den zuvor definierten Schwellen  $t_{min}$  und  $rot_{min}$  verglichen. Dazu wird die euklidische Norm der ermittelten relativen Translation berechnet und überprüft ob diese die Schwelle überschreitet. Bei der Rotation genügt es, wenn eine der Rotationskomponenten die definierte Schwelle überschreitet. Ist mindestens eine der Schwellen überschritten ist für dieses Pose-Paar ein Kartenupdate durchzuführen. Die entsprechenden Indices der Paare werden in einem Array gespeichert, das im Anschluss durchlaufen wird. In Kapitel 4 wurde beschrieben, wie an dieser Stelle Datenassoziationen zwischen der betrachteten Pose  $P_i$  und der TSDF-Karte hergestellt werden können. Zusätzlich stellt das Kapitel heraus, dass eine solche Assoziationsgeneration in vielerlei Hinsicht problematisch und fehlerbehaftet ist. Aus diesem Grund wird an dieser Stelle ein anderer Ansatz verwendet. Wie zuvor eruiert, basiert der hier vorgestellte Ansatz nicht auf einer bereits fertig generierten TSDF-Karte, sondern soll in einen inkrementellen SLAM-Prozess eingegliedert werden. Durch diese neue Grundlage können Anpassungen an der inkrementellen Generation der TSDF-Karte vorgenommen werden. Ein großes Problem der ursprünglichen TSDF-Karte besteht darin, dass in ihr keine relationalen Informationen vorhanden sind. Ein Speichern aller relationalen Daten, wie behandelt in Kapitel 4, durch behandelte Assoziationsbestimmung ist aus Sicht des dafür benötigten Speichers und der dynamischen Anzahl relationaler Daten keine sinnvolle Herangehensweise. Dies wurde in Kapitel ?? allerdings nur angestrebt, um die nicht vorhandenen Umgebungsdaten durch die vorhandene Karte auszugleichen. Dies ist an dieser Stelle nun nicht mehr nötig. Eine Herangehensweise auf der breiteren Datenbasis wäre es, den betroffenen Teil der Karte zu löschen und anhand der vorhandenen Umgebungsdaten in Form der Punktwolken  $\mathbb{C}$  an anderer Stelle neu zu generieren. Problematisch ist hier nur der Schritt des Löschens alter, fehlerhafter Daten aus der Karte. Hierfür werden im Folgenden zwei verschiedene Ansätze vorgestellt und evaluiert. Grundlage hierfür bietet die Erweiterung jeder TSDF-Zelle, bestehend aus einem TSDF-Wert und einem TSDF-Gewicht, um ein weiteres Feld. Dieses beinhaltet den Index der letzten Pose, die in diese Zelle geschrieben hat. Dafür sind nur minimale Anpassungen an der Datenstruktur, sowie dem TSDF-Update vorzunehmen, da nur ein Erstellen des Feldes und die entsprechende Befüllung der Indices im TSDF-Update durchzuführen ist. Eine Zelle, die Default-Werte enthält und bisher nicht durch ein TSDF-Update beschrieben wurde erhält den Index  $-1$ . Durch diese Anpassung wird die Karte aus Sicht des Speichers um 50% vergrößert. Das Ergebnis ist eine diskretisierte Karte die folgende Tupel enthält:

$$\langle value, weight, index \rangle \quad (6.2)$$

Anhand dieser um die Pose-Indices angereicherten Karte kann nun eine Löschung der betroffenen Kartenabschnitte vorgenommen werden. Dazu wird das erstellte Array durchlaufen, welches Informationen zu den Posen enthält, deren Änderungen durch die Graphen-Optimierung in Translation oder Rotation größer ist als die definierten Schwellen. Für jede Pose werden alle TSDF-Zellen ermittelt, deren Pose-Index dem der aktuell betrachteten Pose entspricht und dementsprechend zuletzt von der betrachteten Pose aus beschrieben wurde. Zunächst wurde an dieser Stelle der implementierte Ray-Tracer verwendet, dessen Zentrum auf die derzeit betrachtete Pose gesetzt wird. Schneidet ein Ray eine Zelle, deren Index mit der derzeit betrachteten Pose übereinstimmt, wird sie auf ihren Ursprungszustand zurückgesetzt, indem sie mit Default-Werten

gefüllt wird:  $\langle \tau, 0, -1 \rangle$ . Dieses Verfahren liefert bei bekannten Sensorauflösungen und unter Nutzung der Parameter bei der Generation der Karte gute Ergebnisse. Ist die Sensorauflösung der gegebenen Datensätze unbekannt oder das Sensorsystem durch den simpel aufgebauten Ray-Tracer nicht nachbildbar (wie zum Beispiel bei den Daten des im vorigen Kapitel vorgestellten Hannover-1 Datensatzes) treten ungewünschte Nebeneffekte auf. Dazu gehören zum Beispiel Artefakte in der Karte, die nicht durch den Ray-Tracer gelöscht werden konnten. Aus diesem Grund wurden an dieser Stelle zwei weitere Ansätze entwickelt, die im Folgenden vorgestellt und miteinander verglichen werden. Beide Ansätze teilen den an die Löschung der Daten anschließenden Prozess der Regeneration der Daten basierend auf den Posen des optimierten Graphen. Dieser besteht aus einem iterativen Ablauf der neuen Posen aus den ermittelten Pose-Paaren, in aufsteigender Reihenfolge der Pose-Indices. Für jede neue Pose  $P_i^{new}$  wird beruhend auf der zugehörigen Punktwolke  $C_i$  ein TSDF-Update durchgeführt. Resultat ist eine durch das partielle Update optimierte globale Karte. Die folgenden Abschnitte beziehen sich jeweils auf die Löschung der Teilabschnitte der Karten.

### Reverse-TSDF-Update

Der erste hier vorgestellte Ansatz nutzt die in der erweiterten Datenbasis gegebenen Punktwolken  $C$  zur Identifikation und Löschung der Zellen, die einen identischen Index haben, wie die dazu betrachtete Pose  $P_i$ . Analog zum TSDF-Update werden in diesem Ansatz Rays zwischen der betrachteten Pose  $P_i$  und den Punkten der zugehörigen Punktwolke  $C_i$  generiert. Identisch zum TSDF-Update werden diese Rays nun abgelaufen und die Zellen bestimmt, die im Normalfall durch das TSDF-Update verändert werden. Dazu gehören auch die interpolierten Zellen. Anstelle die Werte der bestimmten Zellen nun so zu manipulieren wie im TSDF-Update, werden die Werte aller Zellen wie oben beschrieben durch Default-Werte ersetzt. Diese Lösung sorgt für die gewünschte partielle Optimierung der Karte, ist allerdings sehr langsam, da für jede Pose ein Shift der lokalen Karte zur aktuell betrachteten Pose notwendig ist. In vorigen Ausführungen wurde bereits beschrieben, dass die Shifts sehr Zeit-intensive Operationen sind. Aus diesem Grund ist die Anzahl an Shifts auf ein Minimum zu reduzieren. Nach diesem Ansatz sind für  $n$  benötigte Karten-Updates  $n$  Shifts für die Löschung der Zellen mittels des Reverse-TSDF-Update und  $n$  weitere Shifts für das grundlegende TSDF-Update durchzuführen. Insgesamt sind dies  $2 \cdot n$  Shifts der lokalen Karte. Besonders bei großen Kartenupdates führt dies zu großen Laufzeiten der Updates. Aus diesem Grund wurde eine weitere Lösung des Problems implementiert, die auf den bisherigen Ausführungen aufbaut. Dies wird im folgenden vorgestellt.

### Global-Map basiert

Wie den vorigen Ausführungen zu entnehmen ist, sollte die Anzahl der Map-Shifts möglichst reduziert werden. Die vorgestellte Abstraktion des Zugriffs auf die globale Karte über eine lokale Karte fester Größe und die dadurch entstehende Notwendigkeit der Verschiebungen stellt allerdings an dieser Stelle ein Problem dar. Eine mögliche Minimierung der Shifts liegt darin, die lokale Karte nur zu verschieben, wenn die aktuell betrachtete Pose eine definierte Distanz in  $x$ ,  $y$  oder  $z$ -Richtung, wie zum Beispiel einem Viertel der jeweiligen Seitenlängen, vom Zentrum der lokalen Karte entfernt ist. Dies reduziert die Anzahl der benötigten Shifts, macht den Ansatz allerdings anfällig für Artefakte, da Zellen mit dem Index der betrachteten Pose nun außerhalb der Begrenzungen der lokalen Karte liegen und somit nicht mehr gelöscht werden können. Aus

Balkendiagramm  
Laufzeit  
Update -> Shift  
vs. restliches  
Update

diesem Grund wird an dieser Stelle ein anderer Ansatz gewählt, der nicht die Abstraktion der lokalen Karte verwendet, sondern direkt auf der globalen Karte arbeitet. Dies reduziert die Anzahl der benötigten Shifts auf 0. Dabei wird wie folgt vorgegangen. Zunächst werden die Namen beziehungsweise Tags aller Chunks der globalen Karte ermittelt. Im Anschluss werden die Chunks nacheinander aus der HDF5 geladen und die enthaltenen TSDF-Zellen durchlaufen. Um zu verhindern, dass für jede Zelle alle Posen durchlaufen und der Index abgeglichen werden muss, wird der Methode ein Array übergeben, das für jede Pose einen Wahrheitswert darüber enthält, ob für die zum Index zugehörige Pose ein Update notwendig ist. Nun ist für die Entscheidung, ob der Inhalt einer Zelle mit Default-Werten überschrieben wird nur eine Abfrage des entsprechenden Array-Elements an der Stelle des in der Zelle enthaltenen Indices von Nöten. Dadurch liegt diese Operation in  $\mathcal{O}(n)$  mit  $n$  als Anzahl der Zellen der globalen Karte. Eine weitere Optimierung dieses Ansatzes wird durch die Vernachlässigung von Chunks, die ausschließlich Default-Werte enthalten, erreicht. Die Anzahl dieser "leeren" Chunks ist stark abhängig von der Größe der lokalen Karte, da bei einer Verschiebung alle neu abgedeckten Chunks zunächst mit Default-Werten initialisiert und danach gegebenenfalls nie wieder beschrieben werden. Je größer die lokale Karte, desto größer die Anzahl leerer Chunks. Im folgenden Abschnitt wird nun das optimierte partielle Update mit dem globalen Update als Base-Line verglichen. Dabei wird sowohl auf die Laufzeit, als auch auf qualitative Unterschiede eingegangen. Die Evaluation erfolgt auf Basis der in Kapitel 5 vorgestellten Datensätze.

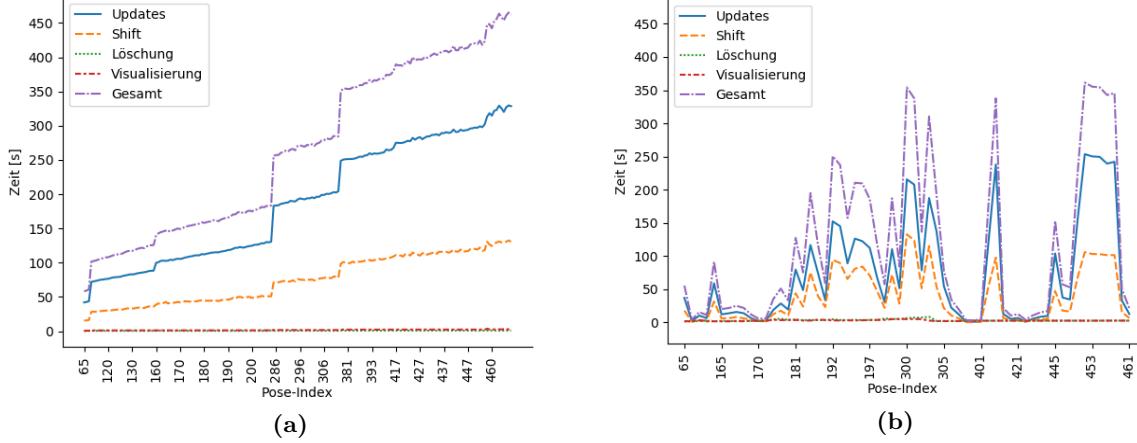
### 6.3 Evaluation der Update-Strategien

In diesem Abschnitt werden die zuvor beschriebenen Ansätze zum Karten-Update eruiert und evaluiert. Zunächst erfolgt ein Vergleich der Laufzeiten der beiden Algorithmen. Abbildung 6.1 vergleicht die Laufzeiten der unterschiedlichen Herangehensweisen an das Problem des Karten-Updates grafisch miteinander. Zu erkennen ist eine deutliche Verbesserung der Laufzeit durch die Einführung des partiellen Updates. Näheres hierzu ist der Unterschrift der Abbildung 6.1 zu entnehmen. D

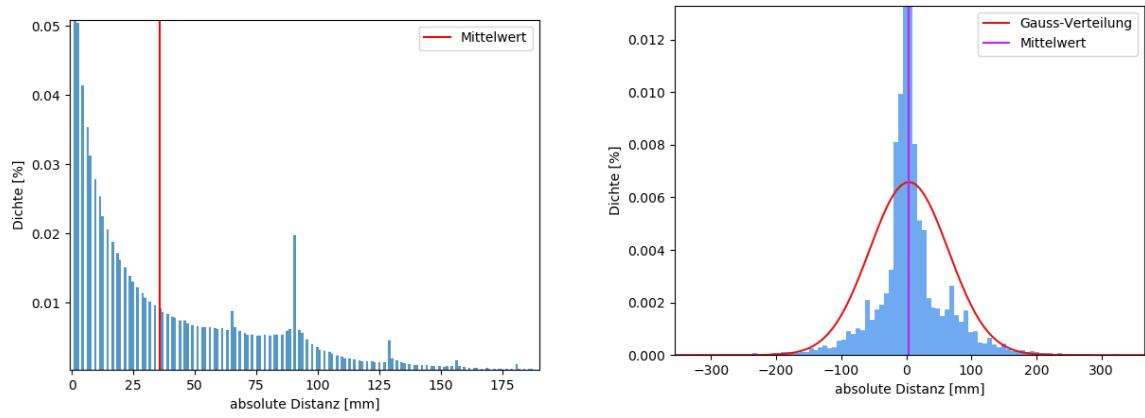
Im Folgenden werden die Ergebnisse des partiellen Updates mit denen des globalen Updates durch einen Vergleich der TSDF evaluiert. Dazu wird ein in [10] implementierter Ansatz zur Rekonstruktion der Oberfläche aus TSDF-Karten in Form eines Dreiecksnetzes verwendet. Dies ergibt sowohl für das globale, als auch für das partielle Update ein Dreiecksnetz bestehend aus *Vertices (Knoten)* und *Faces (Flächen, hier Dreiecksflächen)*. Mit Hilfe der Software Cloud-Compare [8] können die Punktwolken der beiden resultierenden Meshes miteinander verglichen werden. Dieser Vergleich ist in Abbildung 6.2 als Vergleich der berechneten absoluten Distanzen zwischen den Punktwolken dargestellt.

Nachfolgendes Kapitel fasst die Ergebnisse der Arbeit zusammen und liefert einen Ausblick über mögliche zukünftige Arbeiten im Themengebiet dieses Projektes.

Darauf eingehen, dass die Karte nach dem Update nicht identisch zum globalen Update ist, da spezifische Gewichte und Werte im Überlappungsbereich durch einen Wert ersetzt werden, der nicht auf den vorvorigen posen geschriebenen Werten basiert.



**Abbildung 6.1:** Grafischer Vergleich der Laufzeiten des globalen und partiellen Updates auf Basis des vorgestellten Hannover-1 Datensatzes. Dargestellt sind die Laufzeiten für jedes einzelne durchgeführte Update. Zusätzlich wurden kenntlich gemacht, wie sich welche Komponenten der Updates auf die Laufzeit auswirken. Die Laufzeit des globalen Updates ist links, die des partiellen rechts abgebildet. Es ist deutlich zu erkennen, dass die Laufzeit im globalen Update stetig ansteigt. Sprünge in diesem Graphen entstehen durch Teilbereiche des Pfades, in denen keine Schleifen geschlossen wurden und entsprechend kein Update der Karte nötig ist. Im Graphen, der die Laufzeit des partiellen Updates beschreibt sind starke Sprünge zu beobachten. Dies liegt an der variablen Anzahl nötiger Löschungen und TSDF-Updates, basierend auf den angegebenen Schwellen, anhand derer bestimmt wird, ob ein Update durchgeführt wird. Hier betragen diese Schwellen in der Translation 3, 2cm und in der Rotation 2°. In der Gesamtheit benötigt das globale Karten-Update für den Hannover-1 Datensatz mit 468 Posen 13.2h, während das partielle Update 1.96h benötigt. Dies entspricht einer Verringerung der Laufzeit auf 15% der Laufzeit der vorgegebenen Base-Line in Form des globalen Updates. Diese Verringerung gründet sich in der Verringerung der Laufzeiten der einzelnen Bestandteile des Algorithmus durch eine geringere Anzahl an Posen, die an einem Update beteiligt sind. In einigen Fällen reduziert sich diese Anzahl sogar auf 0. Dementsprechend ist vorstellbar, dass in einem optimalen Datensatz mit regelmäßigen Schleifenschlüssen, die nur eine geringe Auswirkung auf den Verlauf des Pfades haben nur sehr wenige oder nahezu keine Updates durchzuführen sind, während beim naiven globalen Update nach jedem Schleifenschluss ein Update durchgeführt wird, ohne Berücksichtigung der genannten Fälle. Die durchschnittliche Zeit, die das Update pro Pose gemäß dieser Daten benötigt, beträgt 0.91s.



(a) Absolute Distanzen n $\ddot{a}$ herster Punkte zwischen den aus dem Dreiecksnetzen des partiellen und globalen Update extrahierten Punktwellen.

(b) Absolute Distanz zwischen den Punktdataen der Punktwolke des partiellen Updates, extrahiert aus dem zugeh $\ddot{o}$ rigen Dreiecksnetz und dem Dreiecksnetz des globalen Updates.

**Abbildung 6.2:** Die hier dargestellten Histogramme zeigen die Verteilungen der absoluten Distanzen zwischen den vorgestellten Methoden zum Update der TSDF-Karte. Verglichen ist jeweils der partielle Ansatz im Vergleich mit dem globalen Ansatz. Die absoluten Distanzen ergeben sich aus der Identifikation der Distanzen der aus den generierten Dreiecksnetzen extrahierten Punktwellen zu den n $\ddot{a}$ chsten Punkten der zu vergleichenden Punktwolke. Bei den Verteilungen handelt es sich nicht um Normalverteilungen, sie  $\ddot{a}$ hneln viel mehr einem exponentiellen Abfall. Aus diesem Grund werden an dieser Stelle nicht die Parameter von Normalverteilungen, sondern lediglich der Mittelwert der Verteilungen berechnet, der als rote vertikale Linie in den Abbildungen dargestellt ist. Im linken Teil der Abbildung ist die Verteilung f $\ddot{u}$ r den in Kapitel 5 vorgestellten Datensatz der Universit $\ddot{a}$ t Osnabr $\ddot{u}$ k dargestellt, im rechten Teil der Abbildung die Verteilung f $\ddot{u}$ r den Hannover-1 Datensatz. Der Mittelwert der linken Verteilung betr $\ddot{a}$ gt 35.89 mm und der der rechten mm. Dies macht deutlich, dass zwar ein Unterschied zwischen den Ergebnissen der betrachteten Update-Verfahren besteht, dieser aber im Schnitt deutlich geringer ist, als die in beiden Datens $\ddot{a}$ tzen verwendete TSDF-Zellgr $\ddot{o}$ ße mit einer Seitenl $\ddot{a}$ nge von 128 mm. Dabei stellt die Zellgr $\ddot{o}$ ße den begrenzenden Faktor f $\ddot{u}$ r die Genauigkeit der Ergebnisse dar.

# Kapitel 7

## Zusammenfassung und Ausblick

### 7.1 Zusammenfassung

### 7.2 Ausblick

In dieser Arbeit wurde bereits versucht Teaser++ zur Verbesserung des Scan-Matchings zu verwenden. Bei der Integration wurde allerdings ein Fehler in der von Teaser++ zur Verfügung gestellten API festgestellt. Augrund dieses Fehlers wurde am GitHub-Repository der Bibliothek ein Bug-Ticket erstellt und die Integration der Bibliothek vorerst hinten an gestellt. Vor Kurzem gab es nun Aktivität am erstellten Ticket und der Fehler scheint behoben. In einer zukünftigen Arbeit lohnt es sich gegebenenfalls, an dieser Stelle anzusetzen und Teaser++ zu integrieren.

Eingehen auf Bottleneck: Map-Shift - $\zeta$  großes Verbesserungspotential durch Parallelisierung  
Eingehen auf TSDF-Update und Reverse Tsdf update (ebenfalls große Verbesserung erwartet).

Schreiben über Path-Exploration + Beispiel

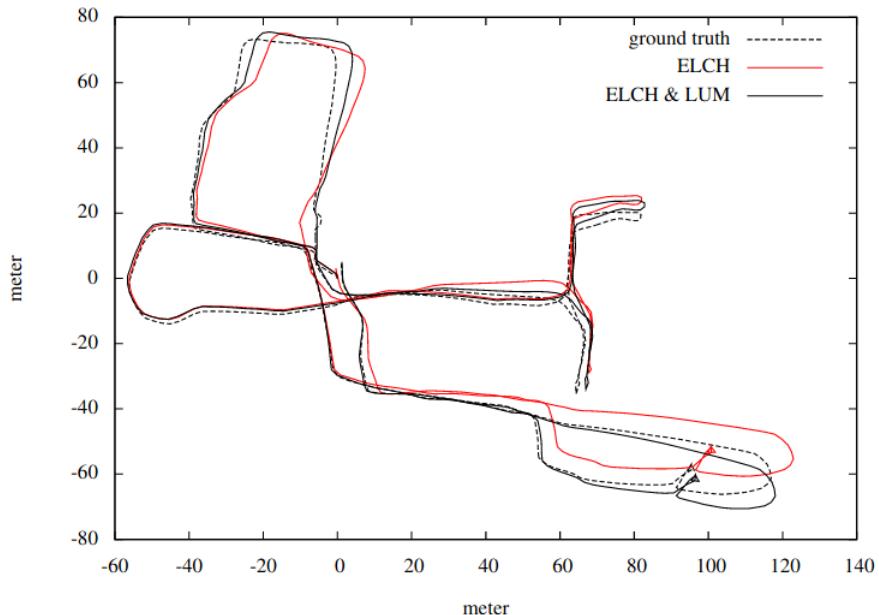
#### 7.2.1 Optimierungen

Wichtig: Optimierung



## **Anhang A**

## **Anhang**



**Abbildung A.1:** Diese Grafik wurde aus [24] entnommen und dient als Vergleich zu den in Kapitel 5 herausgearbeiteten Ergebnissen. Ein direkter, wertebasierter Vergleich mit den Ergebnissen aus [24] ist nicht möglich, da diese dort nur für einen anderen Datensatz angegeben sind. Ein visueller Vergleich der Ansätze zeigt aber eine deutliche Verbesserung des in dieser Arbeit vorgestellten Ansatzes im Gegensatz zum vorgeschlagenen Ansatz aus [24]. Nur an wenigen Stellen ist eine leicht größere Abweichung zur Ground-Truth zu erkennen. An allen anderen Stellen ist eine deutliche Verbesserung erkennbar.

# Literaturverzeichnis

- [1] BENTLEY, Jon L.: Multidimensional binary search trees used for associative searching. In: *Communications of the ACM* 18 (1975), Nr. 9, S. 509–517
- [2] BESL, P.; MCKAY, N.: A Method for Registration of 3-D Shapes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14 (1992), Februar, Nr. 2, S. 239–256
- [3] BESL, Paul J.; MCKAY, Neil D.: Method for registration of 3-D shapes. In: *Sensor fusion IV: control paradigms and data structures* Bd. 1611 Spie, 1992
- [4] BORRMANN, Dorit; ELSEBERG, Jan; LINGEMANN, Kai; NÜCHTER, Andreas; HERTZBERG, Joachim: Globally consistent 3D mapping with scan matching. In: *Robotics and Autonomous Systems* 56 (2008), Nr. 2, S. 130–142
- [5] BRESENHAM, Jack E.: Algorithm for computer control of a digital plotter. In: *IBM Systems journal* 4 (1965), Nr. 1, S. 25–30
- [6] CHEN, Yang; MEDIONI, Gérard: Object modelling by registration of multiple range images. In: *Image and vision computing* 10 (1992), Nr. 3, S. 145–155
- [7] CHETVERIKOV, Dmitry; STEPANOV, Dmitry; KRSEK, Pavel: Robust Euclidean alignment of 3D point sets: the trimmed iterative closest point algorithm. In: *Image and vision computing* 23 (2005), Nr. 3, S. 299–309
- [8] CLOUDCOMPARE: *CloudCompare - home*. <https://www.cloudcompare.org/main.html>. – zuletzt abgerufen am: 16.11.2022
- [9] DELLAERT, Frank: Factor graphs and GTSAM: A hands-on introduction / Georgia Institute of Technology. 2012. – Forschungsbericht
- [10] EISOLDT, Marc; FLOTTMANN, Marcel; GAAL, Julian; BUSCHERMÖHLE, Pascal; HINDE-RINK, Steffen; HILLMANN, Malte; NITSCHMANN, Adrian; HOFFMANN, Patrick; WIEMANN, Thomas; PORRMANN, Mario: HATSDF SLAM – Hardware-accelerated TSDF SLAM for Reconfigurable SoCs. In: *2021 European Conference on Mobile Robots (ECMR)*, 2021

- [11] HDF-GROUP: *THE JDF5 LIBRARY FILE AND FORMAT*. <https://www.hdfgroup.org/solutions/hdf5/>. – zuletzt abgerufen am: 02.11.2022
- [12] HE, Ying; LIANG, Bin; YANG, Jun; LI, Shunzhi; HE, Jin: An iterative closest points algorithm for registration of 3D laser scanner point clouds with geometric features. In: *Sensors* 17 (2017), Nr. 8, S. 1862
- [13] IZADI, Shahram; KIM, David; HILLIGES, Otmar; MOLYNEAUX, David; NEWCOMBE, Richard; KOHLI, Pushmeet; SHOTTON, Jamie; HODGES, Steve; FREEMAN, Dustin; DAVISON, Andrew u.a.: Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011
- [14] LEVENBERG, Kenneth: A method for the solution of certain non-linear problems in least squares. In: *Quarterly of applied mathematics* 2 (1944), Nr. 2, S. 164–168
- [15] LORENSEN, W. E.; CLINE, H. E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: *ACM SIGGRAPH '87*, 1987
- [16] LU, Feng; MILIOS, Evangelos: Globally consistent range scan alignment for environment mapping. In: *Autonomous robots* 4 (1997), Nr. 4, S. 333–349
- [17] MARQUARDT, Donald W.: An algorithm for least-squares estimation of nonlinear parameters. In: *Journal of the society for Industrial and Applied Mathematics* 11 (1963), Nr. 2, S. 431–441
- [18] MCCORMAC, John; CLARK, Ronald; BLOESCH, Michael; DAVISON, Andrew; LEUTENEGGER, Stefan: Fusion++: Volumetric object-level slam. In: *2018 international conference on 3D vision (3DV)* IEEE, 2018
- [19] PRISACARIU, Victor A.; KÄHLER, Olaf; GOLODETZ, Stuart; SAPIENZA, Michael; CAVALLARI, Tommaso; TORR, Philip H.; MURRAY, David W.: Infinitam v3: A framework for large-scale 3d reconstruction with loop closure. In: *arXiv preprint arXiv:1708.00783* (2017)
- [20] RUSU, Radu B.; COUSINS, Steve: 3d is here: Point cloud library (pcl). In: *2011 IEEE international conference on robotics and automation* IEEE, 2011
- [21] SCHULTZ, Jarvis: *tf*. <http://wiki.ros.org/tf>. – zuletzt abgerufen am: 09.10.2022
- [22] SEGAL, Aleksandr; HAEHNEL, Dirk; THRUN, Sebastian: Generalized-icp. In: *Robotics: science and systems* Bd. 2 Seattle, WA, 2009
- [23] SHAN, Tixiao; ENGLOT, Brendan; MEYERS, Drew; WANG, Wei; RATTI, Carlo; RUS, Daniela: Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In: *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)* IEEE, 2020

- [24] SPRICKERHOF, Jochen; NÜCHTER, Andreas; LINGEMANN, Kai; HERTZBERG, Joachim: A heuristic loop closing technique for large-scale 6d slam. In: *Automatika* 52 (2011), Nr. 3, S. 199–222
- [25] WERNER, Diana; AL-HAMADI, Ayoub; WERNER, Philipp: Truncated signed distance function: experiments on voxel size. In: *International Conference Image Analysis and Recognition* Springer, 2014
- [26] WHELAN, Thomas; KAESZ, Michael; FALLON, Maurice; JOHANSSON, Hordur; LEONARD, John; McDONALD, John: Kintinuous: Spatially extended kinectfusion. (2012)
- [27] ZHANG, Ji; SINGH, Sanjiv: LOAM: Lidar odometry and mapping in real-time. In: *Robotics: Science and Systems* Bd. 2 Berkeley, CA, 2014



Endpage für  
Unterschrift  
einbauen