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 Robters 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

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 *Simulatneous Localization and Mapping* (SLAM). Diese Arbeit setzt auf den Konzepten von Eisoldt et al. [9] auf, die einen TSDF basierten, hardware-beschleunigten SLAM Ansatz (*HATSDF-SLAM*) entwickelt und dessen Effizienz und Funktionalität verfiziert 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 könnnen, 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 $T_{i \to j}$ bestimmt und mit der Transformation $T_{i \to j}^{graph}$ 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 akkuratere Karte. Als Basis für die Lösung dieses Problems dient der Ansatz von Lu und Milios [14], die ein Netzwerk aus 2D-Poserelationen erstellen, das sowohl Relationen enthalte, die aus dem Scan-Matching abstammen, als auch Relationen aus Odometrie Messungen enthalte [14]. Borrmann et al. [4] erweitern den Ansatz von [14] auf drei Dimensionen und erstellen einen Pose-Graphen zur Speicherung von Pose-Relationen. Zur Registierung 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. [20] 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 eukldische 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 [20]. [20] verwendet einen Faktor-Graphen zur Speicherung der Pose-Relationen. Hierfür wird die Implementation des Faktor-Graphen von **GTSAM** [8] 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 Borrmann et al. [4] sind dies jeweils die zum Zeitpunkt aufgenommenen Punktwolken. [20] 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 Assozation 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 Eislodt et al. [9], 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. [20] 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 Implementationen sinnvoll beschleunigt werden können, um sie in ein Live Karierungssystem, wie HATSDF-Slam einbauen 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 Datenassozationen zwischen der TSDF-Karte und der Posehistorie generiert werden können. Dies wird evaluiert und die weitere Vorgehensweise disktuiert.

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 Datenassozationen 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

Simulataneous 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 dreidmensionalen Raum **(Posen)** Punktwolken aufgnommen. 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 Registierung 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 Distannzen 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 [11]. Bis heute gibt es zahlreiche Varitationen 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 Scanpunkt und der durch den Modelpunkt und die Oberflächennormale am Modelpunkt beschriebenen Ebene. Dieser Algorithmus wird auch **Normal ICP (NICP)** genannt. Nach [11] reduziert NICP die Anzahl Iterationen und konvergiert schneller gegen eine gewisse Grenze als ICP. Segal et al. [19] 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. [12] nutzen ein TSDF-Voxelgrid und eine Kinect-Tiefenkamera, um Umbegungen zu kartieren, während ein Nutzer die Kinect Kamera durch die Umbebung schwenkt. Whelan et al. [23] optimieren diesen Ansatz durch Nutzung eines Ringbuffers zur Kartierung großer Umgebungen. Im Gegensatz zu genannten TSDF Verfahren, nutzen Eisoldt et al. [9] einen Hardware beschleunigten TSDF basierten SLAM Ansatz, sowie einen 3D-Laserscanner anstelle einer Tiefenkamera. Eisoldt et. al [9] nutzen zur Registierung 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. [21] den Schleifenschluss-Ansatz durch Nutzung einer heuristischen Schleifenschluss-Technik, die im Gegensatz zu bisherigen Methoden einen dünn besetzten SLAM-Graphen nutzen. McCormac et al. [15] schlagen ein Online-SLAM System, welches eine dauerhafte und genaue 3D Karte beliebiger rekonstruierter Objekte darstellt [15]. Die Rekonstruktionen sind als TSDF realisiert. [15] 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. [16] unterteilen den 3D-Raum in starre TSDF-Teilkarten und optimieren die relativen Positionen zwischen diesen. Nach [16] ist eine anschließende Generation der globalen Karte durch ein Zusammenführen der Teilkarten möglich. [16] verwendet für jede Pose des Graphen eine eigene Teilkarte, die in sich konsistent ist. [16] 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. [20] 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

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

3.1 Mathematische Grundlagen

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

3.1.1 Konventionen

Transformationen werden im Folgenden folgendermaßen betitelt:

Konventionen einfügen

Pfad

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

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.

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

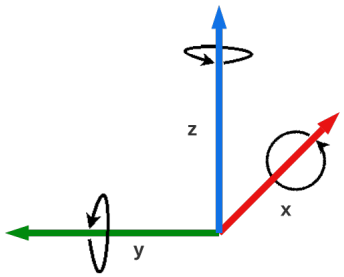


Abbildung 3.1: Schematische Darstellung der Konvention für Koordinatensysteme im ROS Framework. Die z-Achse zeigt nach oben, die x-Achse nach vorne und die y-Achse nach links. Die Rotation um die Achsen ist entsprechend der Konvention im Uhrzeigersinn.

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 untergordneten 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 System.

Auch räumliche Daten wie zum Beispiel Punktenwolken 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 Koordinatensystem-transormation. Diese werden im folgenden Kapitel behandelt.

Transformationen

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

$$f : X \to X \tag{3.1}$$

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

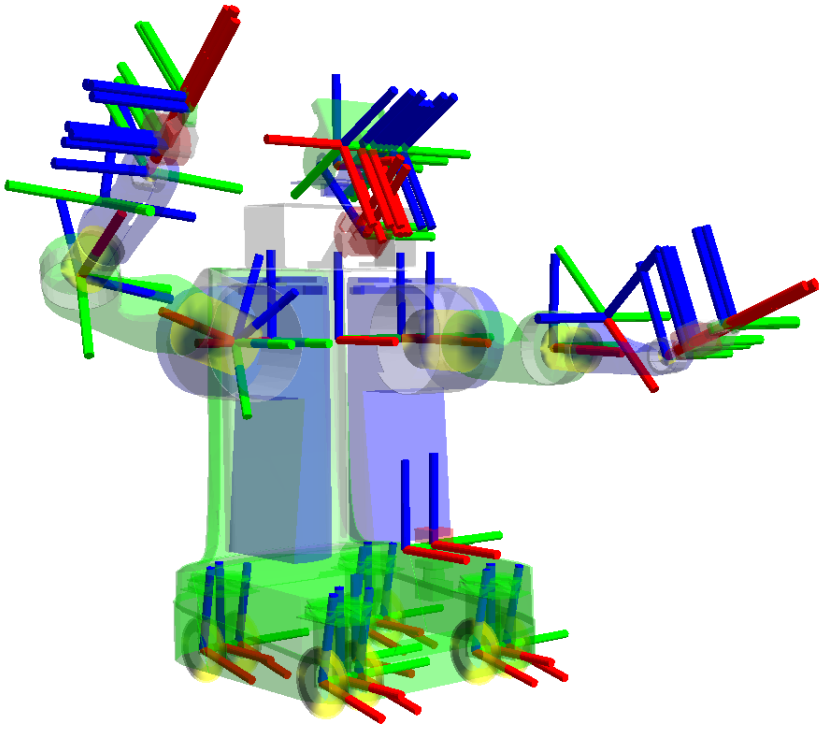


Abbildung 3.2: Schematische Darstellung eines Roboters und dessen beweglicher Teile. Die Ausrichtung der jeweiligen Gelenke wird mit einem lokalen Koordinatensystem beschrieben. Die globale Position einzelner Teile kann durch eine Verkettung der relativen Transformation in Richtung der Wurzel des Baumes bestimmt werden. Bild aus: [18]

C_2 wird die absolute Translation und Rotation beider Koordinatensysteme zum Ursrungskoordinatensystem, wie zum Beispiel den Urpsrung 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 Urpsrungskoordinatensystem.

Es existieren diverse Darstellungsweisen für Koodinatensystemtransformationen. 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.

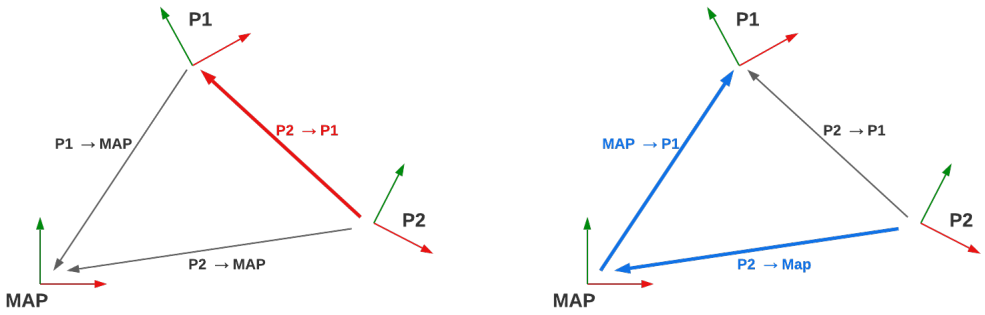


Abbildung 3.3: Schematische Darstellung der bestimmung einer Transformation zwischen zwei Roboter-Posen P_1 und P_2, hier zur Vereinfachung dargestellt in 2D. Gesucht ist die Transformation $(T_{P_2 \to P_1})$, dargestellt im linken Teil der Abbildung als roter Pfeil. Diese kann implizit bestimmt werden durch eine Verkettung der Transformationen $T_{P_2 \to MAP}$ und $T_{MAP \to P1}$, hier dargestellt im rechten Teil der Abbildung in blau. Die Transformation $T_{MAP \to P1}$ ist dabei nicht explizit gegeben. Sie kann berechnet werden durch eine Inversion der Transformation $T_{P1 \to MAP}$. Die finale Gleichung zur Berechnung der relativen Transformation ist dargestellt in Gleichung 3.3.

$$T_{C_X \to C_{MAP}} = \begin{pmatrix} t_x \\ t_y \\ t_z \\ r_x \\ r_y \\ r_z \end{pmatrix} \tag{3.2}$$

Neben der Vektordarstellung kann eine Transformation zusätzlich als eine $4x4$ Matrix dargstellt 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 direkt 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 ($T_{P_2 \to P_1}$) gegeben ist durch:

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

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 **Simulatneous 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** [19] 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 [22]. 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 räumich gesehen hinter dem vom Laser getroffnen 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 Funtion (TSDF)** ist eine Unterklasse der SDF. Sie betrachtet die Distanz zur Oberfläche nur bis zu einer maximalen Distanz $tsdf_{max}$, auch $tau\,(\tau)$ gennant[9]. Alle Werte die weiter von der Oberfäche 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:

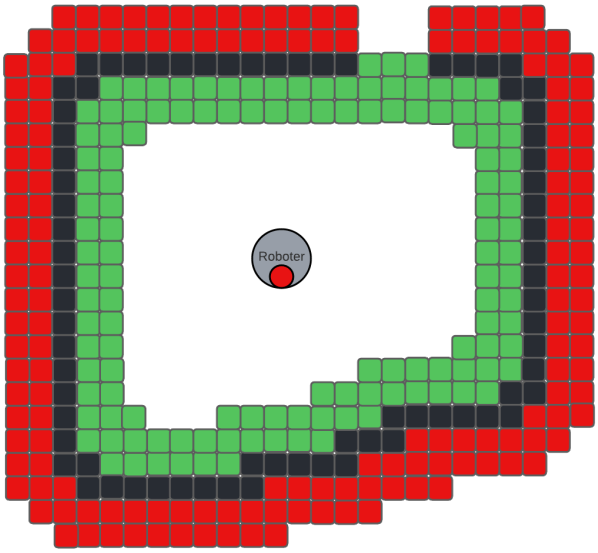


Abbildung 3.4: Schematische Darstellung einer diskretisierten 2D TSDF Karte. Abgebildet sind nur Voxel mit TSDF-Werten im Intervall $]-\tau, \tau[$. In der Mitte der Karte befindet sich ein Roboter mit einem Laserscanner (hier dargestellt in rot). Der Roboter befindet sich zum Beispiel in einem Raum. 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.

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

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} [23] [9]. 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** [13] 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. [9]. 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. [9] basieren ihren SLAM Ansatz auf einer diskreten, inkrementell erweiterten TSDF Karte. Zur Registierung (vergleiche Kapitel 3.3) verwenden sie ebenfalls die TSDF-Karte. Punktwolken werden dabei mit einer **Point-to-TSDF** Strategie an die TSDF Karte registiert [9]. In [9] 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)** [10] 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 serialsierenden Objekte abbilden. Diese Aufteilung in eine globale und eine lokale Karte sorgt dafür, dass [9] 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 Urpsrung der Karte ist im Folgenden als Ursprung des Weltkoordinatensystems definiert.

3.2.2 TSDF-Update

TSDF-Update

erklären.

Nachfolgende Sektion behandelt des 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.

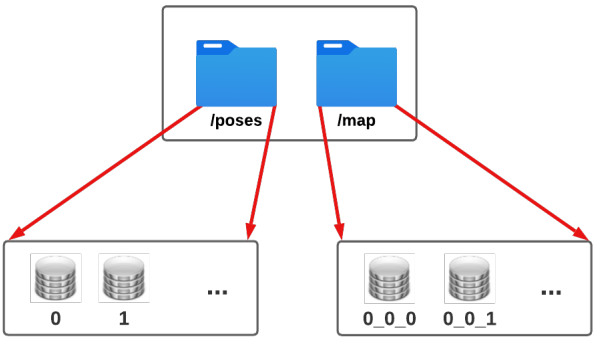


Abbildung 3.5: Innere Struktur der HDF5-Datei, die die globale TSDF-Karte repräsentiert. Auf der obersten hierarchischen Ebene werden die Daten in zwei Gruppen aufgeteilt. */poses* enthält dabei die Trajektorie des Roboters, separiert in einzelne Posen, die mit Null beginnend indiziert sind. */map* enthält 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 **Incremental Closest Points (ICP)** Algorithmus nach Besl und McKay [2] ist ein Algorithmus zur **Registrierung** von Punktwolken. Als **Registierung** 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 registiert 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 Incremental Closest Points** [19] 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 Scanpunktwolke in das Koordinatensystem der Modelpunktwolke. Er gibt dementsprechend an, wie groß die durschnittliche quadrierte Distanz eines Punktes aus der Modelpunktwolke zum euklidisch nächsten Punkt der transformierten Scanpunktwolke 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 δT verändert. Fällt dieses δT 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.

Einzelne Verfahren näher beschreiben, erklären welche Bibiloheken verwendet werden

Varianten (auf die eingegangen wird):

ICP Graph-SLAM -¿ Global Relaxation

1. Grundlagen von SLAM beschreiben -¿ Varianten des SLAM -¿ Bezug zu HATSDF-SLAM 2. Vorraussetzungen (Repräsentationen für Posen (Pfad) und Umgebung) 3. Überleitungen in weitere Sektionen machen -¿ Loop Closure als mögliche Verbesserung des SLAM -¿ TSDF als Kartenrepräsentation -¿ 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 identifizierten, 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 [9] 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 [9] 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 [9] 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.

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

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 einer 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 miteiander 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.

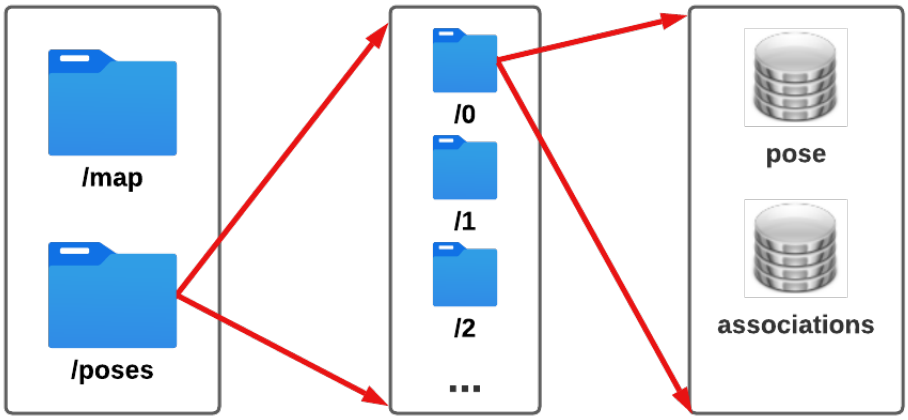


Abbildung 4.1: Schematische Darstellung der HDF5 internen Datenstruktur nach Speicherung der generierten Datenassoziationen zwischen TSDF-Zellen und Posen. Die in Kapitel 4.1 beschrieben *1:N* Beziehung zwischen einer Zelle und den zugehörigen Posen ist hier indirekt realisiert. Anstelle pro Zelle ein 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 das 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:

$$n = vert\_res \cdot hor\_res \tag{4.1}$$

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 setzten sich aus den Öffnungswinkeln zusammen. Der Winkelbereich in horizontaler Richtung beträgt:

$$I_{hor} = [-opening\_degree\_hor, opening\_degree\_hor] \tag{4.2}$$

Der Winkelbereich in vertikaler Richtung beträgt:

$$I_{vert} = [-opening\_degree\_vert, opening\_degree\_vert] \tag{4.3}$$

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
opening_degree	Definiert den vertikalen Öffnungswinkel des Ray-Tracers. Anzugeben in Grad.	45
hor_res	Definiert die horizontale Auflösung des Laserscanners. Der gegebene Wert entspricht der Anzahl *Rays* pro Scanebene.	1024
vert_res	Definiert die vertikale Auflösung des Laserscanners. Der gegebene Wert entspricht der Anzahl an Scanebenen im Laserscan.	128
step_size	Definiert, wie groß die Schrittweite beim Aussenden der einzelnen Rays ist. Der Wert ist in Metern anzugeben. Der Default-Wert ist direkt an die Zellgröße der diskreten TSDF-Karte map_{res} gekoppelt und beträgt $\frac{map_{res}}{2}$.	0.032
ray_size	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:

$$\Delta_{hor} = \frac{opening\_degree\_hor}{hor\_res} \tag{4.4}$$

Die vertikale Schrittweite des Laserscanners beträgt:

$$\Delta_{vert} = \frac{opening\_degree\_vert}{vert\_res} \tag{4.5}$$

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 Δ_{vert} und Δ_{hor} inkrementiert. Aus den beiden Winkeln α und β 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 (α und β gegeben in Radianten, α beschreibt den aktuellen Winkel um die z-Achse, β die aktuelle Rotation um die y-Achse):

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

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 $T_{P_i \to \mathbb{M}}$ ist gegeben durch:

$$\begin{pmatrix} x_{\mathbb{M}} \\ y_{\mathbb{M}} \\ z_{\mathbb{M}} \end{pmatrix} = T_{P_i \to \mathbb{M}} \cdot \begin{pmatrix} x_{P_i} \\ y_{P_i} \\ z_{P_i} \end{pmatrix} \tag{4.7}$$

Auf diese Weise werden alle initialen Endpunkte des emulierten Laserscans berechnet. Auf Basis der berechenten 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:

$$\hat{\vec{r_i}} = \frac{\left\|\vec{r_i} - \vec{t_i}\right\| + step\_size}{\left\|\vec{r_i} - \vec{t_i}\right\|} \cdot \left(\vec{r_i} - \vec{t_i}\right) + \vec{t_i} \tag{4.8}$$

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

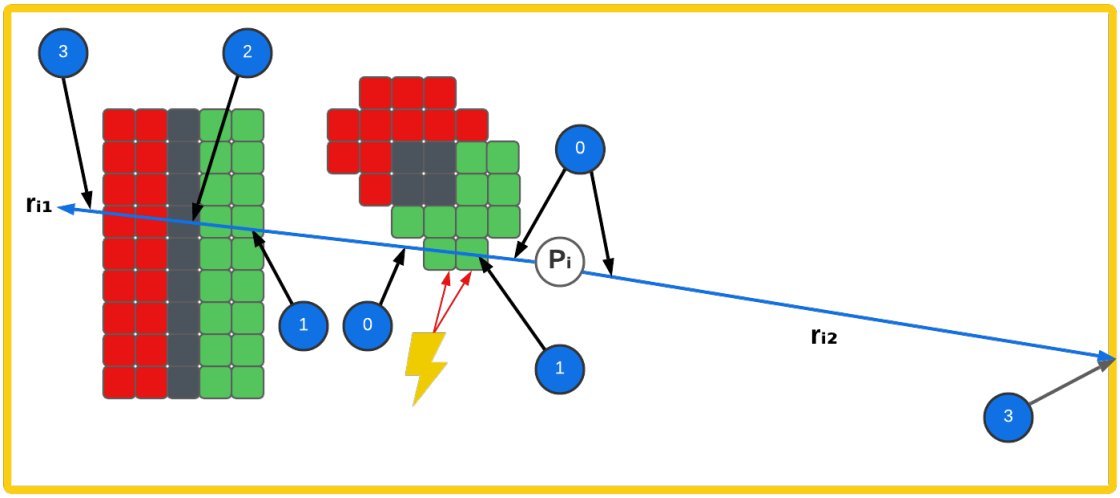


Abbildung 4.2: Schematische Darstellung 2D Darstellung der verschiedenen Zustände eines einzelnen Rays des Ray-Tracers innerhalb einer TSDF-Darstellung. Negative TSDF-Werte dargestellt in rot, positive in grün. Der approximierte Nulldurchgang in der TSDF ist hier gräulich dargestellt, die umgebende lokale 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 Zustandsdisgramm 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 Bedinungen 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 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.

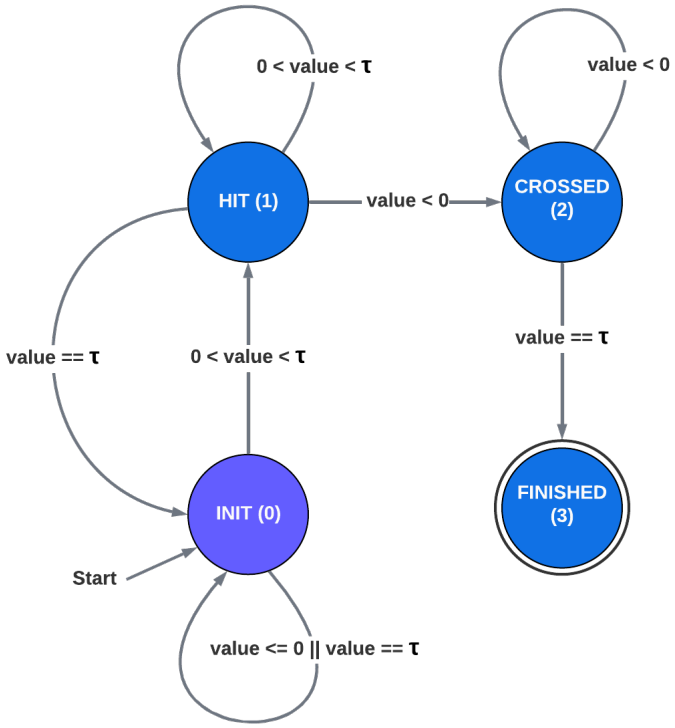


Abbildung 4.3: Zustandsdiagramm der internen Zustände eines Rays. Zustandsübergänge sind beschrieben durch einen initialen Zustand und den TSDF-Wert der aktuellen Zelle (*value*). In Zustand 1 werden gefundene Assoziationen zunächst nicht abgespeichert, da noch nicht bekannt ist, ob diese Zellen zu einem 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 4.5im Vergleich mit den Ergebnisses 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, graphlich 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 usprünglich 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 Fallstricken: Nicht getroffene Zellen, Information loss

ggf. RVIZ-Bild des Ray-Tracing Markers

Tür-Problem

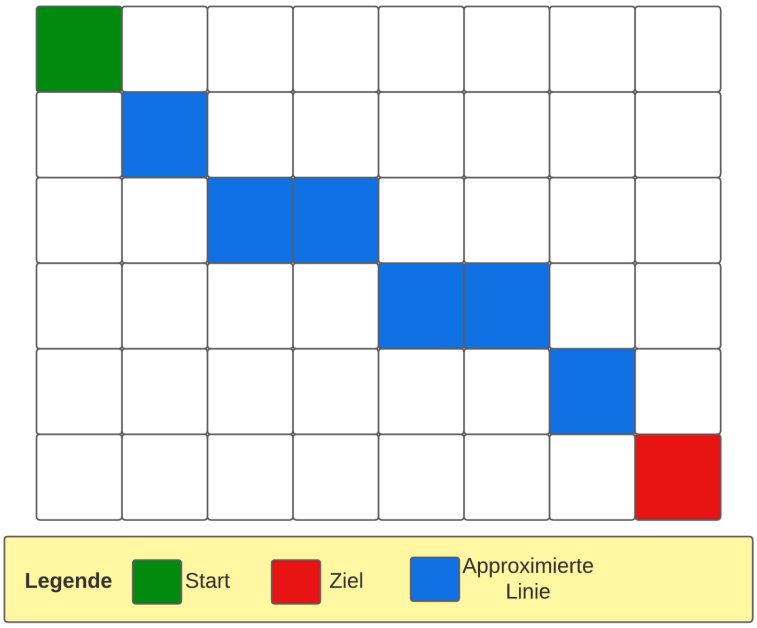


Abbildung 4.4: Schematische Darstellung des Bresenham-Algorithmus in zwei Dimensionen. Der Bresenham Algorithmus bestimmt, welche Voxel die Linie zwischen einem gegebenen Startvoxel und einem Endvoxel am besten beschreiben.

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 $\vec{V_{start}}$, gegeben durch die aktuell betrachtete Pose beziehungsweise Scannerposition P_i und ein Endvoxel $\vec{V_{end}}$. Letzterer berechnet sich aus dem Schnittpunkt des betrachteten Rays mit der Bounding-Box der lokalen Karte, gekennzeichnet durch ihren Ursprung $\vec{p_{lmap}}$ und ihre Seitenlängen $(s_x, s_y, s_z)^T$. . 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])

procedure BRESENHAM($\vec{V_{start}}, map_l, r_i^j$)
  Initialisierung:
  Berechne die Endposition $\vec{V_{end}}$ als Schnittpunkt des Rays r_i^j mit der lokalen Karte map_l
  Berechne Bresenham Parameter (absolute Abstände und Raumrichtungen):
  $dx = \left|\vec{V_{end}}^{\,x} - \vec{V_{start}}^{\,x}\right|$ // *Absolute zwischen den Punkten in alle Raumrichtungen*
  $dy = \left|\vec{V_{end}}^{\,y} - \vec{V_{start}}^{\,y}\right|$
  $dz = \left|\vec{V_{end}}^{\,z} - \vec{V_{start}}^{\,z}\right|$
  $dm = max(dx, dy, dz)$ // *M*aximale Komponente der Manhattan Distanz
  $sx = \vec{V_{start}}^{\,x} < \vec{V_{end}}^{\,x}\ ?\ 1 : -1$ // *Raumrichtung*
  $sy = \vec{V_{start}}^{\,y} < \vec{V_{end}}^{\,y}\ ?\ 1 : -1$
  $sz = \vec{V_{start}}^{\,z} < \vec{V_{end}}^{\,z}\ ?\ 1 : -1$
  Temporäre Vektoren zur Iteration initialisieren:
  $\vec{v_0} = (x_0, y_0, z_0)^T = \vec{V_{start}}$ und $\vec{v_0} = (x_1, y_1, z_1)^T = \left(\frac{dm}{2}, \frac{dm}{2}, \frac{dm}{2}\right)^T$
  for $i = 1; i < dm; i++$ **do** // $(dm-1)$ *mal iterieren*
    Berechnung des nächsten Voxels (gegeben durch $\vec{v_0}$)
    $x_1 = x_1 - d_x;$ if $(x_1 < 0)\,\{x_1 += d_m;\ x_0 += s_x;\}$
    $y_1 = y_1 - d_y;$ if $(y_1 < 0)\,\{y_1 += d_m;\ y_0 += s_y;\}$
    $z_1 = z_1 - d_z;$ if $(z_1 < 0)\,\{z_1 += d_m;\ z_0 += s_z;\}$
    Neues Linien-Voxel gegeben durch: $\vec{v_0}$ bzw. $(x_0, y_0, z_0)^T$
  end for
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 $\vec{V_{end}}$. Ingesamt 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.

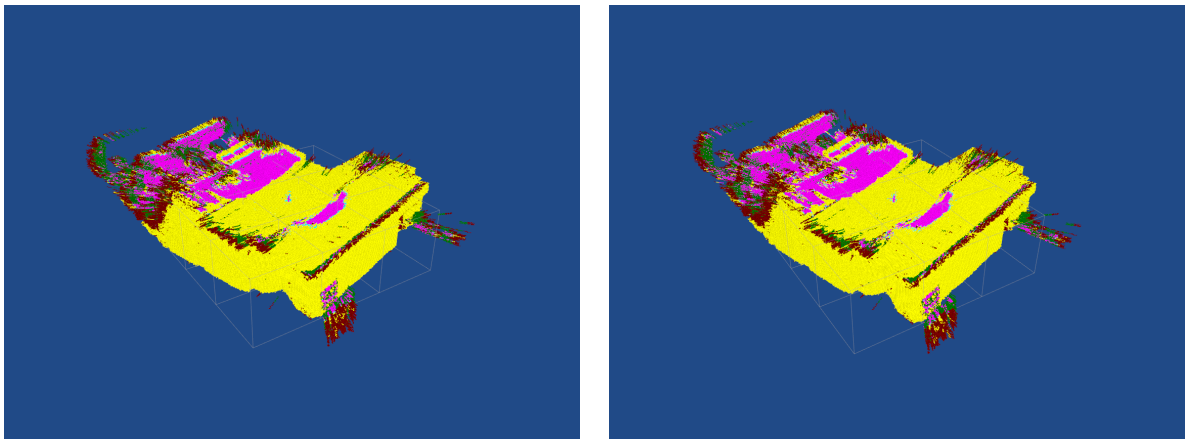
Algorithm 2 Asssoziations-Berechnung mittels Bresenham oder Ray-Tracing

procedure ASSOZIATIONS-BERECHNUNG(P_i, map_{local})
  Initialization:
  Initialisiere die Rays R_i um P_i wie beschrieben in 4.3.1
  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
  Initialisiere einen Zähler für die Anzahl der fertigen Rays: $cnt_{fin} = 0$
  while $cnt_{fin} < size\,(r_{status})$ **do**
    for r_i^j in R_i **do**
      Ermittle nächstes Voxel V_i^j mittels Ray-Tracing oder Bresenham
      Evaluiere V_i^j auf Basis des Zustandsdiagramms in 4.3
      Speichere V_i^j als Assoziation, wenn es gemäß 4.3 mit P_i assoziiert ist
    end for
  end while
  Speichere die gefunden Assoziationen gemäß 4.1 in der HDF5
end procedure

In den folgenden beiden Sektionen 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, das 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 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 gefunden 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 Reflektionen 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

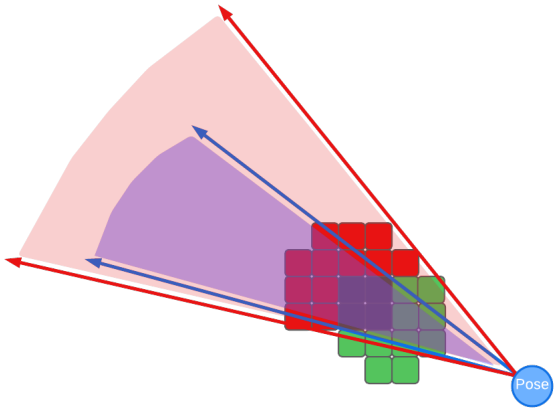


Abbildung 4.6: Diese Abbildung zeigt die Vergrößerung des Verdeckungsbereichs durch die Diskretisierung der Karte als TSDF. Zu sehen ist der ehemalige Verdeckungsbereich des in schwarz gefärbten Objektes. Dieser ist als halbtransparenter blauer Bereich gekennzeichnet. Darum herum zeigt sich der neue 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

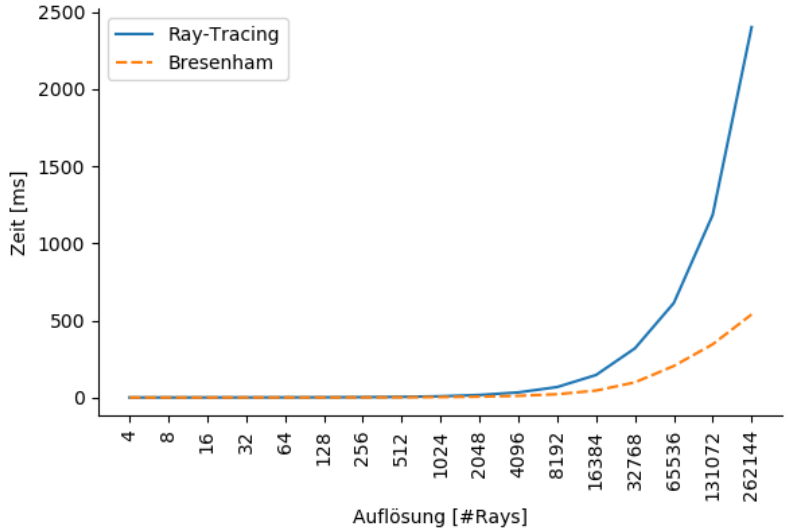
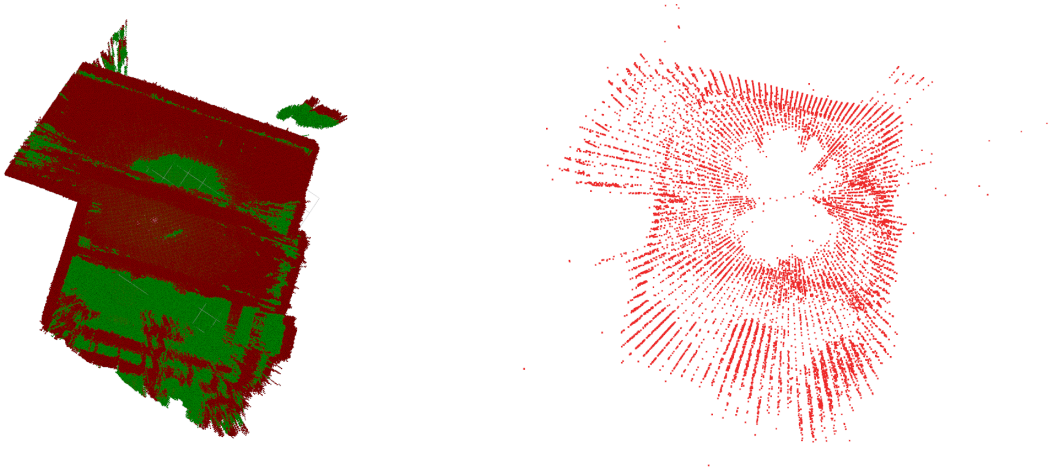


Abbildung 4.7: Laufzeiten der vorgestellten Algorithmen zur Bestimmung der Assoziationen. Es handelt sich bei den jeweiligen Werte um das Mittel der Laufzeiten mehrerer künstlicher Scans beider Algorithmen, ausgehend von ungefähr 30 verschiedenen Positionen. Es ist zu sehen, dass - obgleich der Zeitbedarf 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, 14, 20]. 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.

$$\vec{v_{ij}} = \vec{c_i} + (\vec{c_j} - \vec{c_i}) \cdot \frac{|v_i|}{|v_i|+|v_j|} \tag{4.9}$$

$$\vec{n_{ij}} = (\vec{c_j} - \vec{c_i}) \tag{4.10}$$

Auf Basis der durch $\vec{v_{ij}}$ und 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 Kartenupdate 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 $M = \{C_1, ..., C_N\}$, 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:

$$T_i = \left(T^{opt}_{i \to map}\right)^{-1} \cdot T^{init}_{i \to map} \tag{4.11}$$

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 assoziierte 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 $[P_x, ..., P_{x+n}]$, sowie ein Array aus Pose-Differenzen, gegeben als Transformationsmatrix, $[T_0, ..., T_{N-1}]$ berechnet sich die akkumulierte Zellposition dabei wie folgt:

$$\vec{C_{acc}} = \sum_{j=x}^{x+n} T_j \cdot \vec{C_i} \tag{4.12}$$

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 einfach Transformationen, wie

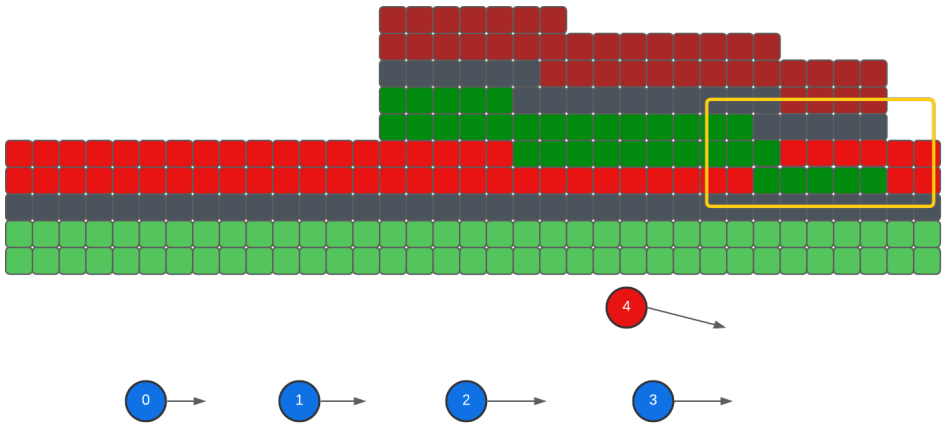


Abbildung 4.9: TSDF-Karte nach Update durch eine fehlerhaft registrierte Pose (Pose 4, dargestellt in rot). Der fehlerhaft hinzugefügte Teil der TSDF-Karte ist farblich durch einen dunkleren Rot- und Grünton hervorgehoben. Eine Assoziationsbestimmung ausgehend von Pose 4 würde hier nicht den fehlerhaften 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 Schleifenschlüssen 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

Loop Closure

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

Dieser Abschnitt befasst sich mit der Detektion von Schleifenschlüssen in einem Pose-Graphen. Die hier vorgestellte Methode kann sowohl in einem inkrementell erweiterten Pose-Grahen, 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.

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, 20] 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 Teilpfades 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 $\mathbb{K} = \{K_0, ..., K_k\}$. 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 $\mathfrak{P}_{init} = \{P_0, ...,_n\}$ zugehörigen Punktwolken $\mathbb{C} = \{C_0, ..., C_n\}$. Für die aktuell betrachte Pose P_{cur} und die zugehörige Punktwolke wird ein Scan-Matching gegen jeden Kandidaten aus $\mathbb{K}$ und dessen zugehörigen Punktwolken C_i durchgeführt und evaluiert. Konvergiert das Scan-Matching mit einer festgelegten Genauigkeit ist der Kandidat validiert. Im Folgenden wird die Punktwolke der aktuell betrachteten Pose P_{cur} als Scan-Punktwolke C_{scan} und die zum zu validierenden Kandidaten zugehörige Punktwolke als Model-Punktwolke C_{model} bezeichnet. Grundlage für das Scan-Matching der beiden Punktwolken ist ein Algorithmus wie **ICP**, **GICP** oder vergleichbare Algorithmen zur Registrierung von Punktwolken. Diese Algorithmen liefern neben einer Information zur Konvergenz in der Regel zusätzlich ein Maß dafür zurück, wie gut die Punktwolken nach der Registrierung aufeinander passen. In dieser Arbeit werden hierzu die Implementationen der Algorithmen der Pointcloud Library [17] verwendet. Diese liefern für beide Algorithmen einen **Fitness-Score (Γ)** zurück, der die durchschnittliche quadrierte Distanz zwischen einem Punkt der Scan-Punktwolke und dem euklidisch nächste Punkt der Model-Punktwolke nach Registrierung der Scan-Punktwolke an die Model-Punktwolke beschreibt. Nachfolgende Formel zeigt diesen Sachverhalt mit s_i als aktuellen Scanpunkt und m_i als zugehörigen, euklidisch nächsten Punkt der Model-Punktwolke und N als Anzahl der Punkte in der Scan-Punktwolke.

$$\Gamma = \frac{\sum_{i=0}^{N} \|\vec{m_i} - \vec{s_i}\|^2}{N} \tag{5.1}$$

Liegt der Fitness-Score unter eine vordefinierten Schwelle Γ_{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 Punktwolkendaten liegen hier jeweils relativ zur Pose, von derer sie aufgenommen wurden, vor. Für die Initialschätzung wird nun die Model-Punktwolke ins Koordinatensystem der Scan-Punktwolke transformiert. Dazu wird zunächst die Transformation $T_{model \to scan}$ über die zugehörigen Posen P_{model} und P_{scan} und die zugehörigen Transformationen $T_{model \to map}$ und

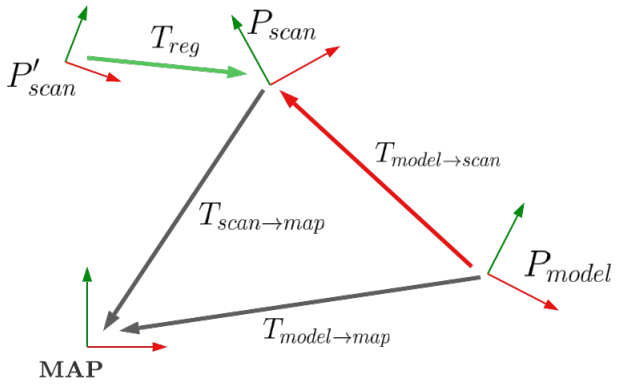


Abbildung 5.1: Diese Grafik stellt die Posen und Transformationen dar, die bei der Detektion eines Schleifenschlusses zur Verwendung kommen. Dabei bestimmt $T_{model \to scan}$ die Vortransformation vom lokalen Model-Koordinatensystem in das des Scans (dargestellt in rot). In grün dargestellt ist die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

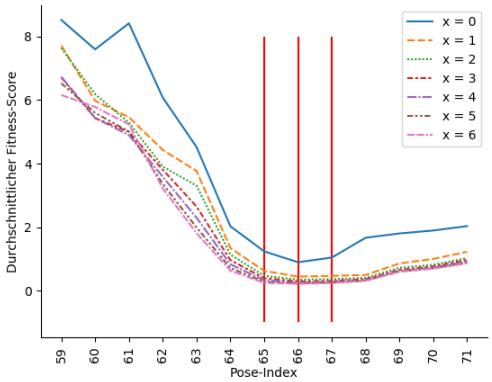
$T_{scan \to map}$ von den Posen ins Ursprungskoordinatensystem bestimmt:

$$T_{model \to scan} = T^{-1}_{scan \to map} \cdot T_{model \to map} \tag{5.2}$$

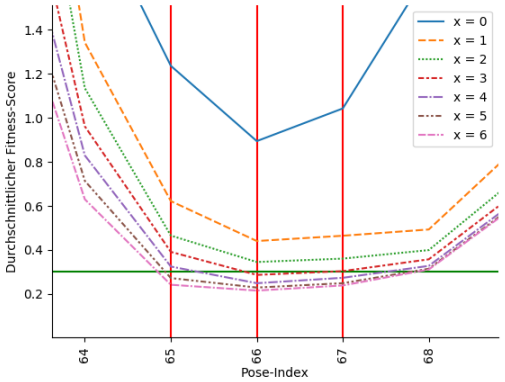
Im Anschluss wird jeder Punkt p^i_{model} der Punktwolke C_{model} mit der bestimmten Transformation $T_{model \to scan}$ transformiert. Nun ist die Model-Punktwolke in das Koordinatensystem der Scan-Punktwolke, gegeben die aktuellen Pose-Schätzungen, vortransformiert. Diese Vortransformation muss nach 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 $T_{scan' \to model}$ die im weiteren Verlauf für die Optimierung des Pose-Graphen benötigt wird ergibt sich wie folgt:

$$T_{scan' \to model} = T^{-1}_{model \to scan} \cdot T_{reg} \tag{5.3}$$

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 mit wenigen Features, fehlerhaft. Hier sind zwar die genannten Rahmenbedingungen erfüllt, allerdings verschlechtern die Schleifenschlüsse das Ergebnis maßgeblich. Um diese Problem zu



(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**.

lösen wurden mehrere Ansätze implementiert und evaluiert. Dazu wurde zunächst das Problem des Scan-Matching näher betrachtet. Dabei stellt sich heraus, dass für ein 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 $\{P_{i-x}, ..., P_i, ..., P_{i+x}\}$ 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 das 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.

Beschreibung der Detektion (bildhaft), Erklärung von Kandidaten und Filterung Beschreibung des optionalen Sichtbarkeitskriteriums

5.2 Graphenoptimierung

Bezug zu GTSAM Library Beschreibung aufnehmen. Wichtigste verwendete Funktionen bennen. Verweis auf GTSAM Paper. Erklärung von Faktorgraphen und Faktoren. Erklärung der Genutzten datenstrukturen und optimizer. Erklärung von noise constraints (Unsicherheiten)

5.3 Optimierungen

1. Vorregistrierung

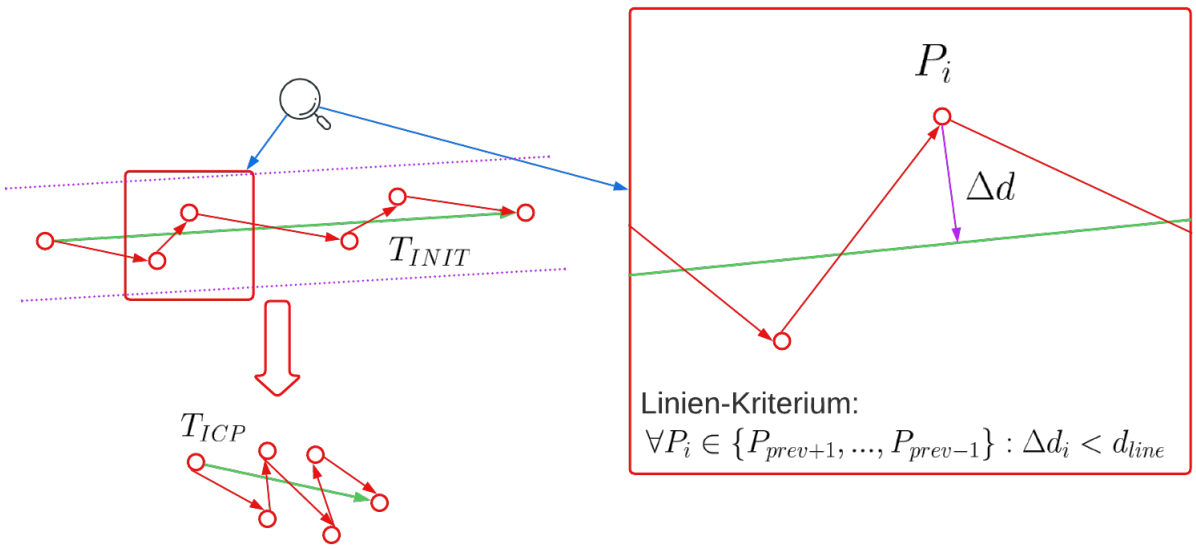


Abbildung 5.3

2. Filtern der Punktwolke

3. Unterschiedliche Scan-Matching Varianten 1. ICP 2. GICP 3. Kurz auf Teaser++ eingehen 4. VGICP

-¿ Analyse des Scan Matchings bezogen auf 1. Vorregistrierung, 2. LC Detektion Matching -¿ Graph über Fitness Counter mit LC Linie (dünn) verschiedene Farben -¿ dazu: starten mit allen varianten und jeweils werte akkumulieren -¿ schreiben in csv

4. Rejectors

einführen: Pointcloud-Enrichment, Grafik: Fitness score im bezug zum enrichment: eine wolke vs. mehrere

5.4 Pseudocode

insert the pseudo code to the whole algorithm

5.5 Datensätze

Verwendete Datensätze und herausforderungen herausstellen

5.5.1 Hannover1

Herausforderungen:

- Datensatz allgemein: falsches Koordindatensystem -¿ Transformation beschreiben

- extrem fehlerbehaftete Rotation in zweitem Kreis (Lösung: Vorregistrierung ICP) (Scan-Matching bekommt extrem divergierte Wolken nicht mehr aufeinander)

- problematisch: fehlerhafte LC Optimierung durch schlechtes Scan Matching (Grund: Feature-armer Flur) -¿ mögliche (noch zu entwickelnde) Lösung: LC's auf Linien gesondert betrachten -¿ Identifikation eines LC auf Linien -¿ Betrachtung der Scan Matching Transformation - wenn aufGeraden kann die Transformation die beiden LC-Punkte nicht vollständig zusammen ziehen

5.5.2 Maps

5.6 Evaluation

Kapitel 6

Map Update

An dieser Stelle ist bereits beschrieben, dass das Map-update auf Basis abgespeicherter Punktwolken durchgeführt wird und keine TSDF-Pose Assoziationen verwendet werden

6.1 Globales Update

6.2 Partielles Update

erste Herangehenweise: ähnlich wie bei globalem Update, aber: nur TSDF für Posen löschen, die um einen Abstand X bzw. einen Winkel Y verschoben wurden. Problem: welche Zellen müssen gelöscht werden, wenn Pose X um δx verschoben wurde? Erste Idee: Ray-Tracing ausgehend von X, getroffene Zellen löschen, die zu einem Update gehören (TODO: wie definiert man das überhaupt?).

Kapitel 7

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 -¿ großes Verbesserungspotential durch Parallelisierung Eingehen auf TSDF-Update und Reverse Tsdf update (ebenfalls große Verbesserung erwartet).

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] Dellaert, Frank: Factor graphs and GTSAM: A hands-on introduction / Georgia Institute of Technology. 2012. – Forschungsbericht

[9] Eisoldt, Marc; Flottmann, Marcel; Gaal, Julian; Buschermöhle, Pascal; Hinderink, 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

[10] HDF-Group: *THE JDF5 LIBRARY FILE AND FORMAT*. `https://www.hdfgroup.org/solutions/hdf5/`. – zuletzt abgerufen am: 02.11.2022

[11] 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

[12] 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

[13] Lorensen, W. E.; Cline, H. E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: *ACM SIGGRAPH '87*, 1987

[14] Lu, Feng; Milios, Evangelos: Globally consistent range scan alignment for environment mapping. In: *Autonomous robots* 4 (1997), Nr. 4, S. 333–349

[15] 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

[16] 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)

[17] Rusu, Radu B.; Cousins, Steve: 3d is here: Point cloud library (pcl). In: *2011 IEEE international conference on robotics and automation* IEEE, 2011

[18] Schultz, Jarvis: *tf*. `http://wiki.ros.org/tf`. – zuletzt abgerufen am: 09.10.2022

[19] Segal, Aleksandr; Haehnel, Dirk; Thrun, Sebastian: Generalized-icp. In: *Robotics: science and systems* Bd. 2 Seattle, WA, 2009

[20] 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

[21] 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

[22] Werner, Diana; Al-Hamadi, Ayoub; Werner, Philipp: Truncated signed distance function: experiments on voxel size. In: *International Conference Image Analysis and Recognition* Springer, 2014

[23] Whelan, Thomas; Kaess, Michael; Fallon, Maurice; Johannsson, Hordur; Leonard, John; McDonald, John: Kintinuous: Spatially extended kinectfusion. (2012)

Endpage für Unterschrift einbauen