Universität Osnabrück
Institut für Informatik
AG Wissensbasierte Systeme

Diplomarbeit

Effizientes Schleifenschließen mit sechs Freiheitsgraden in Laserscans von mobilen Robotern

Die ELCH (Explicit Loop Closing) Heuristik

Jochen Sprickerhof

22. August 2009

Erstgutachter: Prof. Dr. Joachim Hertzberg
Zweitgutachter: Akad. Dir. Klaus Brauer

We shall not cease from exploration
and the end of all our exploring
will be to arrive where we started
and know the place for the first time.

– T. S. Eliot, Four Quartets

Zusammenfassung

Simultane Lokalisierung und Kartenerstellung (SLAM) beschreibt das Problem, mit einem Roboter eine Karte einer unbekannten Umgebung zu generieren und gleichzeitig in ihr zu navigieren. In meiner Diplomarbeit habe ich einen neuartigen Ansatz entwickelt, um das SLAM-Problem mit einem 3D-Laserscanner in sechs Freiheitsgraden zu lösen. Der entwickelte ELCH-Algorithmus muss dabei nicht über alle Laserscans iterieren, um ein konsistentes Ergebnis zu erlangen. Dazu *trennt* er den letzten Scan einer gefahrenen Schleife, registriert ihn neu gegen den Schleifenanfang und verteilt den gefundenen Fehler über einen Posegraphen. Dies ermöglicht eine genaue Kartenerstellung bei einer sehr kurzen Algorithmenlaufzeit.
Um den Ansatz zu testen, habe ich ihn auf zwei Datensätze mit 924 bzw. 468 Einzelscans angewendet und die Ergebnisse mit einer Referenztrajektorie verglichen. Außerdem habe ich den Algorithmus so erweitert, dass er die Datensätze mehrerer Roboter in eine konsistente Karte vereinigen kann.
Ein Teil der Ergebnisse wird gleichzeitig unter [30] veröffentlichen.

Abstract

Simultaneous Localization and Mapping (SLAM) is the problem of building a map of an unknown environment by a mobile robot while navigating this environment using the unfinished map. In my diploma thesis I present a novel approach for solving SLAM using 3D laser range scans with 6 DoF. The developed ELCH algorithm avoids the iteration over all scans to get a consistent result. It *dissociates* the last scan of a loop of acquired scans, reassociates it to the beginning of the loop, and distributes the difference in the pose error over the pose graph. This yields a quality improvement as well as a huge time saving.
I tested my algorithm on two 3D scans of an urban environment with 924 and 468 scans, respectively, and evaluated the results against ground truth. Furthermore I extended it to fuse scans of multiple robots into one consistent map.
Some results will be available in [30], as well.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Mobile Roboter sind dabei einen immer größeren Bereich des menschlichen Lebensraums zu erobern. Ein wichtiger Teilaspekt dabei ist es, die Roboter möglichst autonom agieren zu lassen. Wenn sich ein Roboter zum Beispiel selbstständig durch unbekanntes Gelände bewegen soll, braucht er Informationen über seine Umgebung, also ein Karte. Zur Autonomie gehört auch, dass die Karte nicht vorgegeben ist, sondern vom Roboter selbst aufgenommen und zusammengebaut wird, während er durch die Gegend fährt. Diese Simultane Lokalisierung und Kartenerstellung (Simultaneous Localization and Mapping, SLAM) ermöglicht es, Roboter schnell in neuen und unbekannten Umgebungen einzusetzen. Aber auch in Gegenden, für die eine Karte existiert, ist es oft besser, den Roboter selbst die Umgebung erkunden zu lassen, weil die gegebene Karte nicht genau genug ist oder gar nicht die Information liefert, die der Roboter benötigt. So sind die meisten traditionellen Karten nur zweidimensional. Ein Roboter sollte seine Umgebung aber möglichst komplett erfassen, weshalb für ihn auch die dritte Dimension wichtig ist. Außerdem beinhalten dreidimensionale Karten wesentlich mehr Informationen für den Roboter, z. B. für die Lokalisierung.

SLAM ermöglicht es einem Roboter, eine 3D-Karte seiner Umgebung zu erstellen und sich gleichzeitig in ihr zu lokalisieren. Ohne externe, globale Lokalisierung, wie zum Beispiel GPS, kann der Posefehler dabei mit der Zeit beliebig anwachsen. Schleifenschließen, also das Zurückkehren an bereits besuchte Orte, ermöglicht es dem Roboter, sich in der Karte neu zu lokalisieren und diesen Fehler zu begrenzen. In dieser Arbeit stelle ich einen Algorithmus vor, der Schleifen erkennt, korrigiert, und den gefundenen Fehler schnell über die erstellte Karte verteilt, um eine global konsistente Repräsentation der Umgebung zu erhalten.

In Abbildung 1.1. ist eine Szene vor und nach der Korrektur einer Schleife zu sehen. Links kann man erkennen, dass einige Strukturen doppelt vorhanden sind, zum Beispiel das Gerüst im Vordergrund, oder nicht zusammen passen, wie das Hausdach im Hintergrund. Im rechten Bild sind diese Fehler korrigiert und man kann feinere Strukturen erkennen. Ein wesentlicher Aspekt dabei ist, dass nicht nur die Karte besser wird, sondern auch der Zeit- und Rechenaufwand für notwendige Korrekturen klein bleibt. Ziel ist es dabei, die Algorithmen online und in Echtzeit, also während der Fahrt, auf dem Roboter auszuführen. Mit dem ELCH-Ansatz ist dieses für Fahrten von mehreren Kilometern möglich.

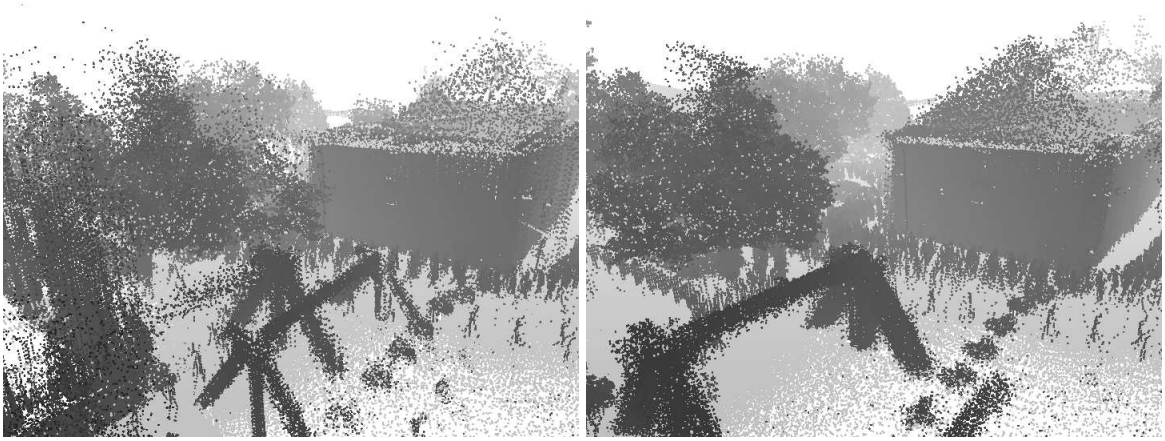


Abbildung 1.1: Eine 3D-Szene vor und nach der Korrektur mit dem ELCH Algorithmus (Szene aus dem Datensatz HANNOVER 2).

1.2 Mobile Robotik

Ausgangspunkt meiner Diplomarbeit ist ein Roboter, bestückt mit einem 3D-Laserscanner wie in Abbildung 1.2. Er erkundet ein Gelände und nimmt, ohne dabei anzuhalten, in regelmäßigen Abständen mit dem Scanner eine 3D-Punktwolke auf. Anhand der Umdrehung der Räder schätzt er dabei seinen ungefähren Aufenthaltsort (Odometrie). Um meine Algorithmen evaluieren zu können, habe ich mich in dieser Arbeit darauf beschränkt, das Programm auf frei verfügbare Datensätze anzuwenden.

1.3 Schleifenschließen in der Literatur

Algorithmen zum Schleifenschließen und Fehlerverteilen in der Literatur lassen sich grob in zwei Typen unterteilen. Auf der einen Seite gibt es direkte, nicht iterative Algorithmen, die einen einmal gefundenen Fehler über die Schleife verteilen. Einfach Ansätze dazu beschreiben Andreas Nüchter und Kai Lingemann in [22], wo sie den Fehler gleichmäßig über die Schleife verteilen, und in [23], wo sie die Streckenlänge zwischen zwei Scans als Gewichtung nutzen. Dieser Ansatz funktioniert allerdings nicht mehr bei mehreren, ineinander verschlungenen Schleifen. Der ELCH-Ansatz ist als eine direkte Erweiterung dieser Idee zu sehen.

Weiter gibt es Ansätze, einen erweiterten Kalman-Filter (EKF) zum Schleifenschließen zu nutzen. Ein Kalman-Filter fusioniert Mittelwert und Varianz einer Poseschätzung, in der Robotik wird meistens die Odometrie verwendet, mit einer Messung, wozu man das Ergebnis des Scanmatchings nutzt. In [2] und [8] verwenden Bailey und Durrant-Whyte den EKF, um Landmarken zu verfolgen, sodass der Zustandsvektor eine Karte repräsentiert. Schließt der Roboter nun eine Schleife, so wird dies anhand einer wiedergefundenen Landmarke erkannt und sowohl die Roboterpose als auch alle Landmarkenkoordinaten im Zustandsvektor werden korrigiert. Solange man sich im 2D befindet, also zwei Translations- und einen Rotationsfreiheitsgrad hat, funktioniert diese Methode recht gut,

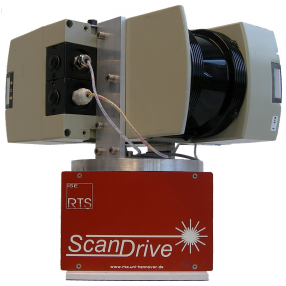


Abbildung 1.2: Links: Der Roboter Erika der Universität Hannover, der zum Erstellen der Datensätze in Kapitel 4 genutzt wurde (Bild aus [32]). Rechts: Der Laserscanner der Universität Hannover.

da alle Variablen linear unabhängig voneinander sind. Eine Erweiterung auf sechs Freiheitsgrade ist hingegen nicht ohne weiteres möglich, da der EKF nur linear zwischen der Poseschätzung und der Messung interpoliert. Bei größeren Rotationen wird dies aber ungenau, da alle drei Rotationswinkel nichtlinear voneinander abhängen. In Abschnitt 3.2 gehe ich näher auf das Problem ein und schildere meine Lösung für ELCH.

Trotzdem ist EKF-SLAM (Thrun et al. [31]), also SLAM mit dem Kalman-Filter, sehr weit verbreitet und es gibt viele Veröffentlichungen, die sich mit dem Erkennen und Wiederfinden von Landmarken beschäftigen. Nieto et al. teilen in [19] Laserscans in Cluster ein, um darin herausragende Punktkonstellationen zu suchen. Dazu wird ein *object saliency score* genannter Index gebildet, welcher sich aus der Kovarianzmatrix eines Scans berechnet. Liegt dieser Index oberhalb einer Schranke, so wird das Clustersegment als neue Landmarke in den Zustandsvektor des EKF eingetragen. Alternative Methoden von Newman et al. [17, 18] arbeiten mit Kamerabildern, in denen markante Punkte gesucht werden.

Ein zweiter Ansatz basiert auf Graphen, in denen der Fehler einer Schleife verteilt wird (Graph-SLAM). Hierbei werden meistens alle Posen, die dicht genug beieinander liegen, durch Kanten im Graphen verbunden. Schleifen werden geschlossen, indem iterativ durch einen Gradientenabstieg die Pose mit dem kleinsten Fehler im Graphen gesucht wird. Die einfachste Möglichkeit dabei ist, das Problem auf ein lineares Gleichungssystem zu beschränken, wie es von Lu und Milios in [16] für 2D und von Borrmann und Elseberg in [4] erweitert auf 3D oder auch von Gutmann und Konolige in [12, 14] gemacht wird. Olson et al. vergleicht in [25] hierfür verschiedene Verfahren zum Lösen des Gleichungssystems und stellt ihnen einen eigenen, hybriden Ansatz entgegen. Bei ihm wird die Methode der kleinsten Quadrate mit einem stochastischen Gradientenabstieg kombiniert [24]. Der Fehler wird dabei in einem Spannbaum verteilt, welcher aus dem SLAM-Graphen berechnet wurde. Erweitert auf 3D findet eine Abwandlung dieses Verfahrens auch bei Grisetti et al. in [11] Anwendung. In Abschnitt 2.6 gehe ich näher auf die Unterschiede zu meiner Methode ein.

Ansätze, SLAM in Echtzeit auf dem Roboter auszuführen, wurden für verschiedene Verfahren entwickelt. Für Graph-SLAM beschreiben Folkesson und Christensen in [9] eine Methode, um den Graphen

zu verdünnen, damit nur lokale Änderungen vorgenommen werden müssen. Dazu wird für die Knoten eine Energiefunktion ausgewertet, die sich aus der Wahrscheinlichkeit für die Korrektheit der Scanpose bildet. Knoten, deren Energie minimal ist, werden aus dem Graphen entfernt und die restlichen in Richtung ihres Minimums verschoben. Dieser Vorgang wird iteriert, bis ein globales Minimum erreicht ist. Einen alternativen Ansatz beschreibt Freese in [10], bei dem die SLAM-Karte in kleine Abschnitte unterteilt wird. Die einzelnen Teile werden in einem Baum verwaltet und der Schleifenfehler stückchenweise darin verteilt. Das Verfahren basiert auf Wahrscheinlichkeitsverteilungen und kann mit sechs Freiheitsgraden arbeiten. In [26] wird von Paz ein divide & conquer Ansatz vorgeschlagen, um einen erweiterten Kalman-Filter in linearer Laufzeit zu erhalten.

1.4 Wissenschaftlicher Beitrag

Die in dieser Arbeit beschriebene ELCH-Strategie hat mehrere Vorteile gegenüber den im vorherigen Abschnitt beschriebenen Verfahren. Im Gegensatz zu den Graph-SLAM-Ansätzen besteht der ELCH-Graph nur aus der Trajektorie und einer Kante pro geschlossener Schleife. Er hat damit wesentlich weniger Kanten als z. B. die von Borrmann, Grisetti und Olson in [4, 11] und [24] beschriebenen Verfahren und kann damit größere Aufgaben schneller lösen. Zusätzlich ist er, anders als die eben genannten Methoden, nicht iterativ. Einen gefundenen Fehler verteilt er direkt im Graphen, statt in einem Spannbaum wie Grisetti und Olson [11, 24]. Zur Fehler- und Kovarianzberechnung nutze ich bekannte und viel getestet Algorithmen (ICP), welche aber ohne größere Änderungen an der Strategie austauschbar sind. Damit ergibt sich ein Framework, in dem sich auch neue Ideen testen lassen. Die von mir in [21] implementierte Version ist dabei auf sechs Freiheitsgrade optimiert, prinzipiell ist die Strategie aber auf eine beliebige Anzahl von Dimensionen anwendbar.

2 Grundlagen

Diese Arbeit erweitert das SLAM 6D-Programm von Andreas Nüchter und Kai Lingemann, welches unter [21] zu finden ist. Der grobe Ablauf des bisherigen Programms ist in Abbildung 2.1 zu sehen. Die einzelnen Teile werde ich in den folgenden Abschnitten beschreiben.

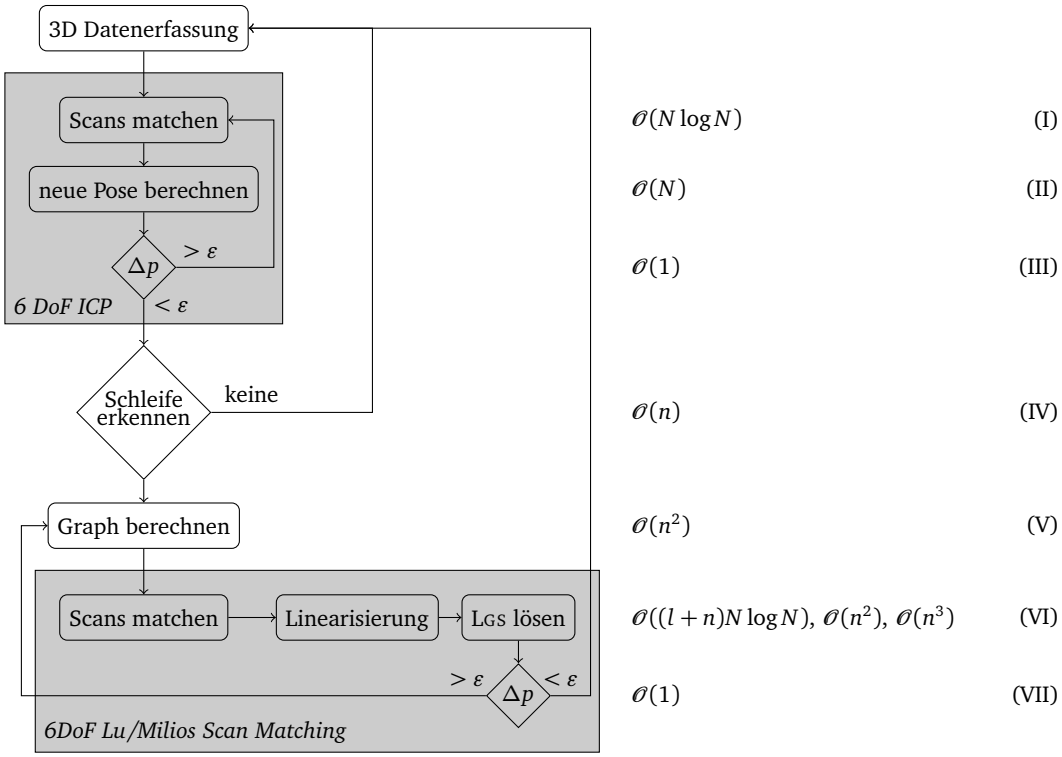


Abbildung 2.1: Der bisherige Programmablauf des SLAM-Programms mit ICP (I bis III) und LUM (VI und VII). Δp ist die Änderung der Posen durch ICP bzw. LUM und bildet das Abbruchkriterium für beide Algorithmen. Auf der rechten Seite stehen die Laufzeiten der einzelnen Programmfunktionen in $\mathcal{O}$-Notation. Dabei ist N die Anzahl der Punkte in einem Scan, n die Anzahl der Scans und l die Anzahl der Schleifen im Datensatz.

2.1 ICP-Algorithmus

Der Iterative Closest Point-Algorithmus (ICP, siehe Besl und McKay, [3]) registriert Daten in einem Modell. Registrieren bedeutet, dass eine Transformation (Translation **t** und Rotation **R**) berechnet wird, die alle Daten im globalen Koordinatensystem des Modells verortet, sodass die Fehlerfunktion

$E(\mathbf{R}, \mathbf{t})$ (Posefehler) minimiert wird. Zu zwei gegeben, sich teilweise überlappenden, 3D-Scans, werden zunächst V Punktkorrespondenzen mit einem kD-Baum gesucht. Diese gehen als Datenpunkte $\mathbf{d}_i$ und Modellpunkte $\mathbf{m}_i$ in die Minimierungsfunktion ein:

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{V} \left\| \mathbf{m}_i - (\mathbf{R}\mathbf{d}_i + \mathbf{t}) \right\|^2 .$$

Da die Punktkorrespondenzen approximativ sind, wird das Verfahren mit der neuen Transformation wiederholt, bis sich die Fehlerfunktion nicht mehr ändert. Das gefundene Minimum ist dabei normalerweise nicht global, da selbst zwei von exakt der gleichen Position aufgenommene Scans nicht die gleichen Datenpunkte enthalten. Die Transformation nach der letzten ICP-Iteration liefert aber normalerweise eine recht gute Schätzung für die Pose des Roboters.

Korrespondenzen finden. Zum Finden der Punktkorrespondenzen werden alle Punkte des Modells in einem kD-Baum repräsentiert. Dies ermöglicht ein schnelle und speicherplatzeffiziente Suche der Punktpaare. Jedem Datenpunkt des neuen Scans wird dabei der Punkt mit dem kürzesten Abstand im Modell zugeordnet. Dieses Verfahren wird als *Matching* bezeichnet.

Fehler berechnen. Für die Berechnung der Fehlerfunktion gibt es verschiedene Implementierungen. Exemplarisch stelle ich die SVD-Methode nach Arun et al. [1] vor, welche auch von Hertzberg et al. in [13] beschrieben wird. Hierzu wird zunächst die Rotation aus der Korrelationsmatrix $\mathbf{H}$ bestimmt und dann eine passende Translation berechnet. Um $\mathbf{H}$ zu bestimmen, werden als erstes die Schwerpunkte der Punktmengen ermittelt:

$$\mathbf{c}_m = \frac{1}{N} \sum_{i=1}^{N} \mathbf{m}_i .$$

Dann werden die Abstände aller Punkte zum Schwerpunkt berechnet (für $\mathbf{d}_i'$ ergeben sich die Formeln analog):

$$\mathbf{m}_i' = \mathbf{m}_i - \mathbf{c}_m .$$

Die Matrix $\mathbf{H}$ ergibt sich nun aus der Multiplikation der verschobenen Punktmengen:

$$\mathbf{H} = \sum_{i=1}^{N} \mathbf{m}_i' \mathbf{d}_i'^T .$$

Mit der Singulärwertzerlegung (SVD) bekommt man nun drei Matrizen $\mathbf{H} = \mathbf{U}\Sigma\mathbf{V}^T$, aus welchen mit $\mathbf{R} = \mathbf{V}\mathbf{U}^T$ die Rotation berechnet wird. Abschließend kann die Translation mit Hilfe der Schwerpunkte bestimmt werden:

$$\mathbf{t} = \mathbf{c}_m - \mathbf{R}\mathbf{c}_d .$$

Wenn man alle bisherigen Scans zu einem Metascan vereint, kann ICP auch Schleifen schließen und den globalen Fehler minimieren. Dies ist aber sehr rechenaufwendig und dauert lange, da der Algorithmus über alle Scans und alle Punkte in diesen iterieren muss.

2.2 Schleifen finden

Um eine Schleife zu schließen, muss man diese zunächst finden. Ohne globale Lokalisierung kann dieses Problem beliebig komplex werden. Da dies aber nicht Teil meiner Diplomarbeit ist, nutze ich nur einen sehr simplen Algorithmus, um eine Schleife zu erkennen (siehe auch Ausblick in Kapitel 5). Dazu wird während des Registrierens des i-ten Scans die Distanz zu allen vorherigen Scans berechnet. Fällt diese unter eine gegebene Minimaldistanz (z. B. 5 m), geht das Programm von einer möglichen Schleife aus. Wenn der Abstand eines folgenden Scans wieder oberhalb der Minimaldistanz liegt, wird die Schleife als geschlossen angesehen und ein Loop Closing Event ausgelöst. Außerdem werden die beiden Scans mit dem kürzesten Abstand als Anfang und Ende der Schleife ermittelt. Damit dabei nicht aufeinander folgende Scans als Schleife erkannt werden, kann eine minimale Schleifenlänge vorgegeben werden (z. B. 20 Scans).

Diese Heuristik ist sehr einfach gehalten und versagt, wenn die initialen Poseschätzungen aus der Odometrie zu schlecht sind, oder das Registrieren per ICP zwei Scans falsch aufeinander zieht. Dabei kann es sowohl passieren, dass eine Schleife nicht gefunden wird, da die Minimaldistanz nicht unterschritten wird, als auch, dass fälschlicherweise eine Schleife gefunden wird. Das Problem kann man durch komplexere Algorithmen umgehen, die z. B. Merkmale in Scans finden und diese zuordnen. Wenn eine globale Lokalisierung gegeben ist (z. B. GPS), können auch diese zur Schleifenerkennung genutzt werden, da hier die beschriebenen Probleme nicht auftreten. Bei den in Kapitel 4 benutzten Datensätzen tauchen diese Problem nicht auf, wenn für die Parameter Minimaldistanz und Schleifenlänge sinnvolle Werte gesetzt werden.

2.3 Kovarianzen

Die Kovarianz zwischen zwei Laserscans lässt sich über die Mahalanobis-Distanz abschätzen. Dazu werden, wie beim ICP-Algorithmus, Punktpaare in beiden Scans gesucht. Die Mittelwerte und Differenzen bilden Schätzungen für den Erwartungswertvektor und Abstand beider Scans, sodass sich die Kovarianz berechnen lässt. Dorit Borrmann und Jan Elseberg haben dies in ihrer Bachelorarbeit [4] für Eulerwinkel und Quaternionen hergeleitet und implementiert, sodass ich auf die zur Verfügung stehenden Methoden zurückgreifen konnte.

2.4 Strategie von Lu und Milios

Lu und Milios beschreiben in [16] eine globale Optimierungsstrategie für 2D-Scans. Diese wurde von Dorit Borrmann und Jan Elseberg in [4, 5] auf 3D-Scans ausgebaut. Das Verfahren gliedert sich in vier Teile. Zunächst werden alle Scans identifiziert, die sich möglicherweise überlappen, und in einem Graphen gespeichert. Dann werden diese Scans gematcht, um die Kovarianzen zu bestimmen, und die gefundene Transformationen in eine Matrix eingetragen. Durch Linearisierung erhält man ein Gleichungssystem, dessen Lösung die neuen Scanposen bildet. In der SLAM-Software sind mehrere Varianten implementiert, die die Rotation verschieden behandeln. Da die Version mit Eulerwinkeln (LUM 1) in der Software am meisten getestet wurde und stabil läuft, habe ich mich

in dieser Arbeit auf sie beschränkt. Um nicht nur einen Vergleichsalgorithmus zu haben, stehen in Abschnitt 4.1.3 auch die Laufzeiten für die Berechnung mit Quaternionen (LUM 2).

2.5 Dijkstra-Algorithmus

Die ELCH-Strategie nutzt den Dijkstra-Algorithmus, um den Fehler im Graphen zu verteilen. Dieser ist ein Standardverfahren, um kürzeste Wege in einem Graphen zu finden [6, 7]. Als Eingabe bekommt er einen Graphen mit nicht negativen Kantenkosten und einen Startknoten. Zurück gibt er ein Vorgängerarray, das zu jedem Knoten im Graphen den Vorgängerknoten auf dem kürzesten Weg zum Startknoten enthält. Im Programm nutze ich die Dijkstra-Implementierung der Boost Graph Library [28], da sie sehr schnell und frei verfügbar ist.

2.6 Schleifenschließen – Stand der Technik

Bisherige Ansätze zum Schleifenschließen (siehe auch Abschnitt 1.3) in Graphen von Olson [25, 24] und Grisetti [11] errechnen immer erst einen Spannbaum und verteilen den Fehler darin. Um zu zeigen, welche Probleme dabei entstehen können, habe ich einen 2D-Testdatensatz erstellt. In Abbildung 2.2 stehen auf der linken Seite die unkorrigierten und auf der rechten Seite die korrigierten Graphen. In Graph (a) und (c) wurde dabei jeweils eine Kante weggelassen (grau), um die Schleife *C-D-E-H-I-J* zu brechen und aus dem gesamten Graphen einen Spannbaum zu machen. Korrigiert man in diesen Graphen die Schleife von *A* bis *G* (in (b) und (d)), so werden die Knoten *I*, *J* und *H* verschoben, ohne die zusätzliche Kante zu berücksichtigen. Deswegen passen die Kanten danach nicht mehr zum restlichen Graphen. Der in dieser Arbeit vorgestellte Algorithmus ELCH kann mit beliebigen Graphen, wie in Abbildung (e), umgehen, was eine bessere Korrektur (f) ermöglicht.

Ein zweiter Unterschied besteht in den hinzugefügten Kanten. Während bei ELCH nur eine Kante pro gefundener Schleife zum Graphen hinzugefügt wird, werden sowohl bei Olson und Grisetti, als auch bei Borrmann et al. [5] überall dort, wo der Abstand zwischen zwei Posen klein genug ist, sodass eine Kovarianz zwischen den Scans berechnet werden kann, neue Kanten eingefügt. Der resultierende Graph hat daher erheblich mehr Kanten als der von ELCH, wie man auch in Abbildung 2.3 sehen kann. Dies hat wesentliche Auswirkungen auf die Geschwindigkeit der Berechnungen im Graphen.

Als letzter großer Unterschied sei angemerkt, dass alle eben genannten Verfahren iterativ arbeiten und in jedem Schritt nur einen Teil des Fehlers verteilen. Durch das wiederholte Durchlaufen von Fehlerberechnen und -verteilen nähern die Algorithmen sich asymptotisch dem optimalen Ergebnis. Die ELCH-Strategie ist dagegen nicht iterativ, sondern verteilt nur einmal den gefundenen Fehler.

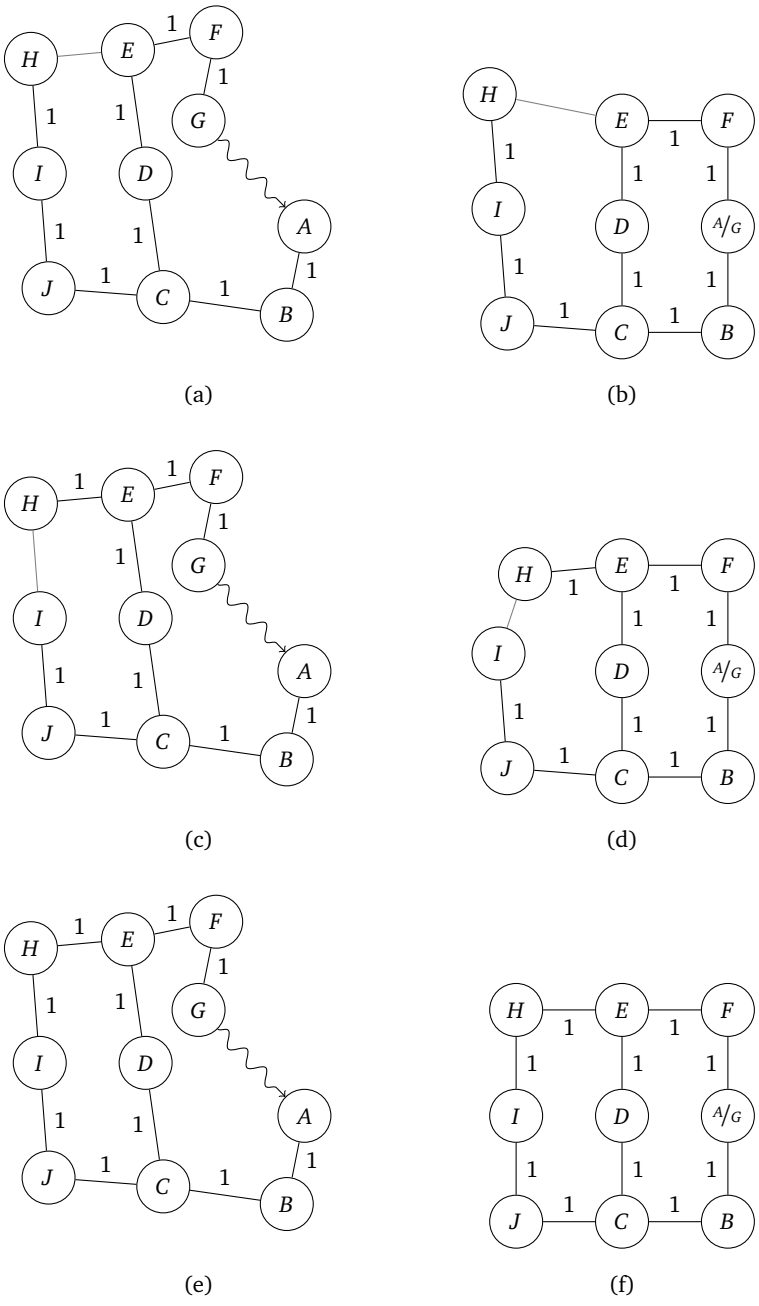


Abbildung 2.2: 2D Graphen (nur Translationsfehler) und die mit dem ELCH-Ansatz korrigierten Versionen (rechts), wobei jeweils die Schleife zwischen *A* und *G* geschlossen wird. Bei (a) und (c) wurde eine Kante (grau) weggelassen um aus den Graphen einen Spannbaum zu machen, was zu einer schlechteren Positionierung der Knoten *H*, *I* und *J* führt.

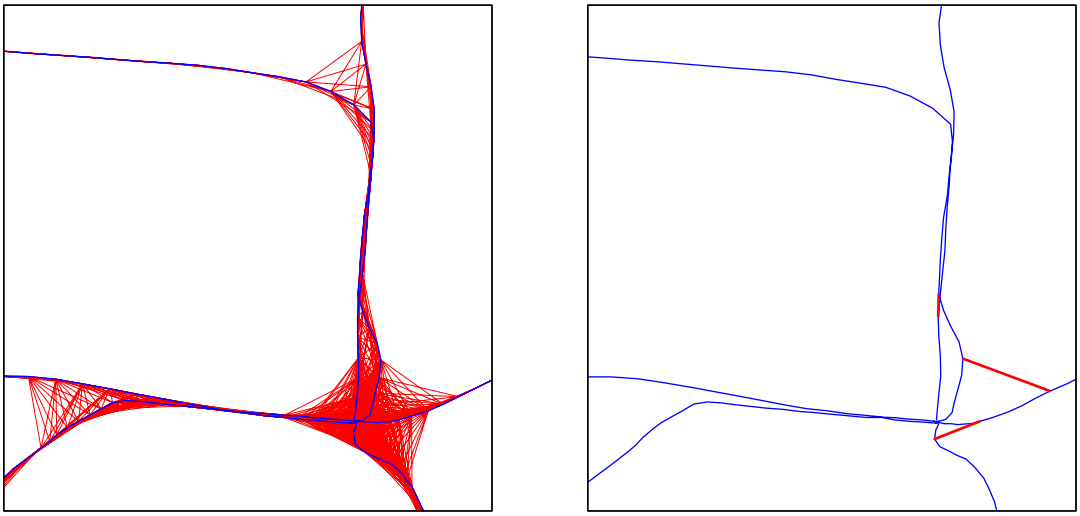


Abbildung 2.3: Links: Ein typischer SLAM-Graph bei Olson, Grisetti und LUM [5, 11, 24, 25] mit den durch die Trajektorie verbundenen Knoten (blaue Kanten) und allen weiteren matchbaren Scans (rote Kanten). Rechts: Der SLAM-Graph von ELCH, in den nur eine Kante pro geschlossener Schleife eingefügt wird.

3 Die ELCH-Strategie

Alle bisher besprochenen Verfahren haben entweder nur den lokalen (ICP) oder den globalen Fehler (LUM) minimiert. Globale Verfahren haben den Nachteil, dass sie mit steigender Kartengröße sehr viel Rechenzeit benötigen. Die Explicit Loop Closing Heuristik (ELCH) geht einen Mittelweg zwischen beiden. Sie nutzt den ICP-Algorithmus, um den akkumulierten Fehler der Roboterfahrt zu bestimmen. Dabei setzt sie mehrere Scans zu einem Metascan zusammen, um ein besseres Matching zu bekommen. Den gefundenen Fehler verteilt ELCH anhand einer Heuristik über die Trajektorie. Eine abschließende globale Korrektur durch LUM kann die Fehler der Heuristik korrigieren, braucht aber sehr viel weniger Iterationen als ohne ELCH, da die Scans vorher schon fast in ihrer endgültigen Pose liegen.

3.1 Herleitung

ELCH macht sich die Eigenschaft zu Nutze, dass bei einer Schleife der Anfang und das Ende etwa auf dem selben Punkt liegen. Wenn man also feststellt, dass man an einer Position schon einmal war, kann man den Fehler, der sich seit der ersten Begegnung eingestellt hat, berechnen und damit explizit angeben. Diesen Fehler gilt es dann möglichst gut über die gefahrene Strecke zu verteilen. Dazu nutze ich einen ungerichteten SLAM-Graphen, in dem jede Roboterpose mit Laserscan einem Knoten und der Weg dazwischen einer Kante entspricht. Zusätzlich wird für jede geschlossene Schleife genau eine Kante zwischen dem Anfangsknoten v_f und dem Endknoten v_l eingefügt (siehe Abbildung 2.3 rechts).

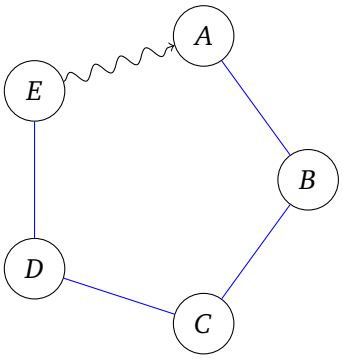
In diesem Abschnitt werde ich zunächst zwei Kantentypen definieren, um damit alle möglichen Kanten in einem Graphen kategorisieren zu können.

Definition (Schleifenkante)**.** Eine Kante $\{v_\alpha, v_\beta\}$ in einem Graphen $G = (V, E)$ bezüglich zweier Knoten v_f und v_l sei definiert als *Schleifenkante*, wenn es zwei knotendisjunkte Wege $(v_i, \ldots, v_f)$ und $(v_j, \ldots, v_l)$ gibt, mit $i, j \in \{\alpha, \beta\}$ und $i \neq j$.

Definition (Abzweigungskante)**.** Eine Kante $\{v_\alpha, v_\beta\}$ in einem Graphen $G = (V, E)$ bezüglich zweier Knoten v_f und v_l sei definiert als *Abzweigungskante*, wenn kein Weg $(v_i, \ldots, v_f)$ knotendisjunkt mit einem Weg $(v_j, \ldots, v_l)$ ist, für $i, j \in \{\alpha, \beta\}$ und $i \neq j$.

Satz. *Eine Kante in einem zusammenhängenden Graphen $G = (V, E)$ mit zwei Knoten v_f und v_l ist entweder eine Schleifenkante oder eine Abzweigungskante.*

Beweis. Eine Schleifenkante kann nicht gleichzeitig Abzweigungskante sein, da sie zwei knotendisjunkte Wege besitzt. Eine Abzweigungskante kann nicht gleichzeitig Schleifenkante sein, da sie keine zwei knotendisjunkten Wege zu v_f und v_l besitzt. Eine Kante kann nicht weder Abzweigungskante noch Schleifenkante sein, da der Graph dann nicht mehr zusammenhängen ist. □



Knoten	Gewicht
A	0
B	$1/4$
C	$2/4$
D	$3/4$
E	1

Abbildung 3.1: Eine einfache Schleife von A bis E und die durch den Loop Optimizer-Algorithmus (LOA, Abschnitt 3.3) ermittelten Gewichte. Alle Kanten sind Schleifenkanten (blau).

Anders ausgedrückt wird jeder Pfad vom Start- zum Endknoten nur Schleifenkanten enthalten. Dabei hängt es vom Start- und Endknoten ab, ob eine Kante Schleifenkante oder Abzweigungskante ist. Diese Definitionen nutze ich nun, um anhand von eindimensionalen Beispielen Lösungen zur Fehlerverteilung herzuleiten, bevor ich in Abschnitt 3.3 einen Algorithmus dafür vorstellen werde.

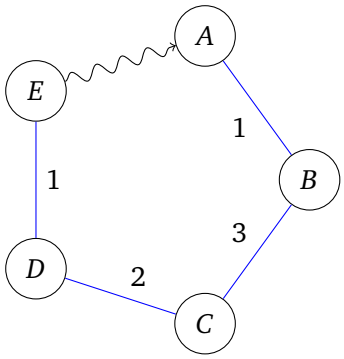
3.1.1 Eine Schleife

Gegeben sei eine Schleife wie in Abbildung 3.1. Dabei bildet der Knoten A den Ursprung und Knoten E soll wieder mit ihm übereinstimmen. Auf der Fahrt über B, C und D soll sich der skalare Fehler δ akkumuliert haben, welcher durch die Schlangenlinie dargestellt wird. Es ist klar, dass Knoten E genau um 1δ verschoben werden muss, während Knoten A im Ursprung bleibt, also um 0δ verschoben wird. Die übrigen Knoten gilt es nun um einen Bruchteil von δ zu verschieben, also um $w\delta$ mit $w \in [0, 1]$. Wenn keine Information darüber vorliegt, wo der Fehler gemacht wurde, ist es sinnvoll, ihn gleichmäßig zu verteilen. Die daraus resultierenden Gewichte w stehen rechts in der Tabelle von Abbildung 3.1.

3.1.2 Eine Schleife mit Kantengewichten

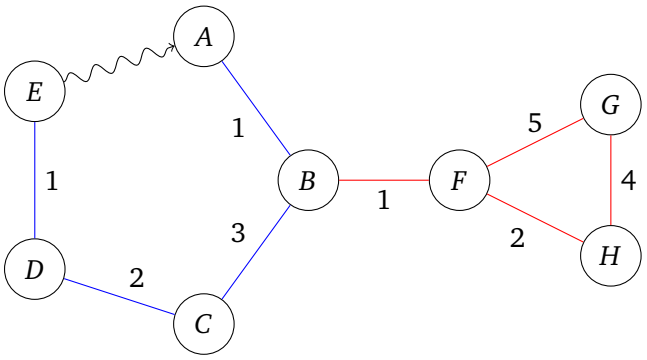
Sind geeignete Kantengewichte bekannt, kann man diese nutzen, um die Knotengewichte zu berechnen. Die Kantengewichte entsprechen dabei dem Fehler, der auf der Fahrt entlang der Kante gemacht wurde. Eine einfache Schätzung dafür sind die Diagonalelemente der Kovarianzmatrix aus Abschnitt 2.3, also die Varianzen. Hat man, wie in Abbildung 3.2, (skalare) Varianzen c_i für die Kanten gegeben, kann man damit den Fehler verteilen. Das Gewicht für den s-ten Scan ergibt sich dann aus folgender Formel:

$$w_s = \frac{\sum_{i=1}^{s} c_i}{\sum_{i=1}^{n} c_i}.$$



Knoten	Gewicht
A	0
B	$1/7$
C	$4/7$
D	$6/7$
E	1

Abbildung 3.2: Um Kantengewichte erweiterter Graph aus Abbildung 3.1 und die sich dadurch veränderten Knotengewichte.



Knoten	Gewicht
A	0
B	$1/7$
C	$4/7$
D	$6/7$
E	1
F	$1/7$
G	$1/7$
H	$1/7$

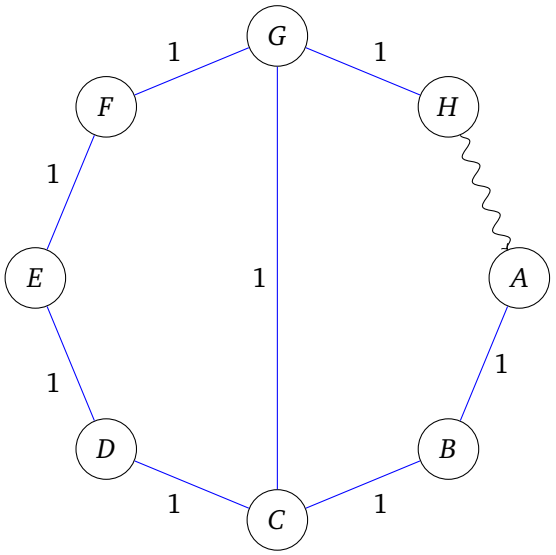
Abbildung 3.3: Graph einer Schleife (blau) mit zusätzlicher Kante an einem Knoten (Abzweigung, rot). Die berechneten Knotengewichte gleichen denen aus Abbildung 3.2, wobei die angehängten Knoten F, G und H die selben Gewichte wie Knoten B bekommen.

3.1.3 Eine Schleife mit einer Abzweigung

Abbildung 3.3 zeigt den gleichen Graphen wie Abbildung 3.2, nur dass Knoten B um ein Anhängsel erweitert wurde. Da die Knoten F, G und H nur durch eine Abzweigungskante mit dem Hauptkreis verbunden sind, bekommen sie das gleiche Gewicht wie Knoten B. Sie werden also bei der ELCH-Transformation mit B mitgezogen. Alle zusammenhängenden Abzweigungskanten nenne ich eine Abzweigung.

3.1.4 Zwei Schleifen

Komplizierter wird es nun, wenn es zwei alternative Schleifen in der Trajektorie gibt (Abbildung 3.4). Damit keiner der beiden Wege zerstört wird, müssen beide passend verschoben werden. Den Graphen kann man dabei zunächst wie eine einfache Schleife betrachten, in dem man den Weg mit der kleineren summierten Varianz wählt. Dies ergibt Sinn, da eine kleinere Gesamtvarianz auch ein genaueres Verteilen des Fehlers ermöglicht. Der übrig gebliebene Weg muss nun so korrigiert werden,



Knoten	Gewicht
A	0
B	$1/4$
C	$2/4$
D	$9/16$
E	$10/16$
F	$11/16$
G	$3/4$
H	1

Abbildung 3.4: Zwei Schleifen. Zunächst wird die kürzere Schleife *A*-*B*-*C*-*G*-*H* geschlossen und dann werden die Gewichte von D, E und F angepasst.

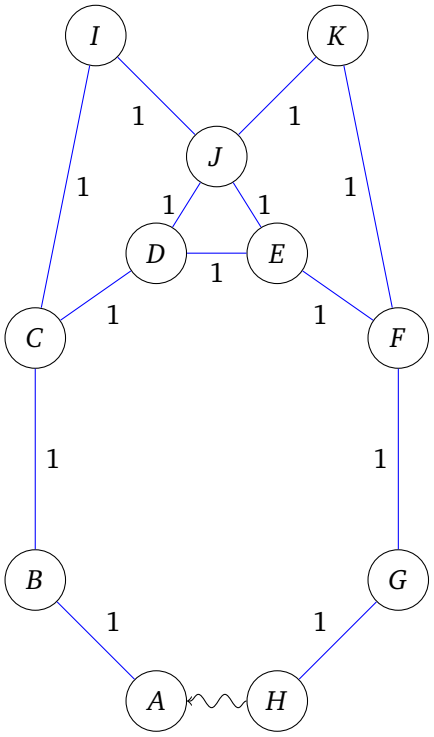
dass er wieder nahtlos an die kleinere Schleife passt. Dazu bietet sich das selbe Vorgehen wie bei der ursprünglichen Schleife an, nur mit neuem Anfang C und Ende G.

Ein beliebiger, zusammenhängender Graph, mit einer neu zu schließenden Schleife, lässt sich nun durch die Unterteilung in Schleifenkanten und Abzweigungskanten so aufteilen, dass jedem Knoten ein Gewichtungsfaktor zugeordnet wird, der, multipliziert mit δ, die Verschiebung des Knotens bestimmt.

3.1.5 Fledermaus

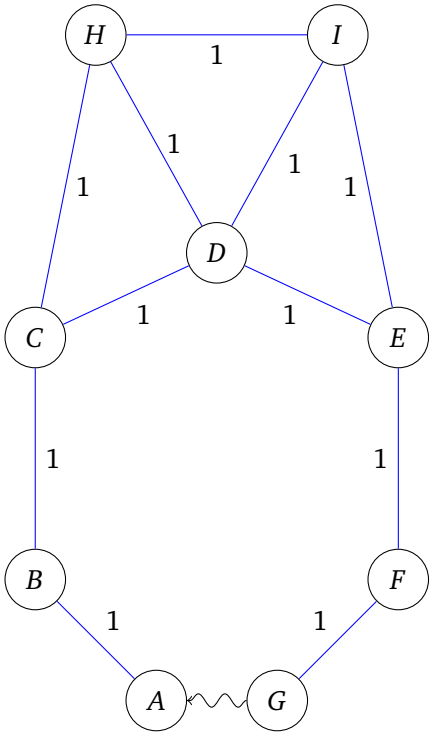
Die Abbildungen 3.5 und 3.6 zeigen Graphen mit komplizierteren Strukturen, auf die sich die Beispiele aus den vorhergehenden Kapiteln anwenden lassen. In Abbildung 3.5 wird zunächst eine Schleife zwischen A und H geschlossen, an der die zusätzlichen Knoten I, J und K hängen. Diese bilden zwei ineinander verschlungene Schleifen, die „Fledermausohren“. Durch die Kantengewichtung werden sie allerdings aufgebrochen und als erstes die kleine Schleife *D*-*J*-*E* korrigiert. Danach entstehen zwei Einzelschleifen *C*-*I*-*J* und *J*-*K*-*F*, welche die Knotengewichte für I und K eindeutig definieren.

Dass Knotengewichte nicht immer eindeutig sind und nicht immer alle Kanten berücksichtigt werden, kann man in Abbildung 3.6 erkennen. Dies ist kein Problem, da es sowieso nur einen Fehler zu verteilen gibt, zusätzliche Kanten also nur die Gewichtung ändern könnten. Am Hauptkreis A bis G hängen die beiden Knoten H und I, welche jeweils mit drei Kanten mit dem restlichen Graphen verbunden sind. Für diese gibt es nun jeweils drei Möglichkeiten die Gewichte zu berechnen. Beginnt man mit H, ist über die Schleife *C*-*H*-*D* ein eindeutiges Gewicht von $w_H = 5/12$ gegeben. Für I bestehen nun die Schleifen *D*-*I*-*E*, *H*-*I*-*D* und *H*-*I*-*E* mit den jeweiligen Gewichten $7/12$, $11/24$ und $13/24$. Eine mögliche Vorgehensweise wäre es nun, alle diese Gewichte zu berechnen und in geeigneter Weise einen Mittelwert zu finden. Dies würde allerdings der Idee einer schnellen Heuristik



Knoten	Gewicht
A	0
B	$1/7$
C	$2/7$
D	$3/7$
E	$4/7$
F	$5/7$
G	$6/7$
H	1
I	$11/28$
J	$7/14$
K	$17/28$

Abbildung 3.5: Fledermaus bestehend aus einem Hauptkreis mit den Knoten A bis H und zwei, ineinander verschlungenen, Mausohren C-I-J-E und D-J-K-F und die dafür berechneten Gewichte.



Knoten	Gewicht
A	0
B	$1/6$
C	$2/6$
D	$3/6$
E	$4/6$
F	$5/6$
G	1
H	$5/12$
I	$7/12$

Abbildung 3.6: Alternative Fledermaus mit entsprechenden Gewichten

widersprechen. Schaut man sich die Gewichte für I genauer an, so stellt man fest, dass sie nie kleiner als das von H und nie größer als das von E werden. Für die Heuristik ergibt sich deswegen einfach aus der Reihenfolge der Knoten ein eindeutiges Gewicht (siehe Abschnitt 3.3).

3.2 ELCH mit sechs Freiheitsgraden

ELCH für sechs Dimensionen funktioniert fast so wie in einer Dimension. Die drei Translationsfreiheitsgrade x, y und z kann man als linear unabhängig voneinander ansehen. Ein Fehler in der Schätzung für die x-Koordinate wirkt sich nicht auf die y-Koordinate aus. Um einen Translationsfehler zu korrigieren, berechne ich für jede Koordinate eine eigene ELCH-Korrektur, welche zusammengenommen den Transformationsvektor ergeben. Für eine Rotation ist dies nicht unbedingt möglich, da die einzelnen Winkel linear abhängig sind. Das Problem lässt sich auf mehrere Arten lösen, welche von der Darstellung der Rotation abhängen.

3.2.1 Eulerwinkel und Lineare Interpolation

Eulerwinkel sind die einfachste Art, eine Rotation im dreidimensionalen Raum darzustellen. Für jede Koordinatenachse wird ein Winkel angeben, um den diese gedreht werden soll. Dabei verschiebt eine Drehung um eine Achse die anderen beiden Achsen. Eulerwinkel sind also nicht kommutativ und man muss eine Reihenfolge festlegen. Dies macht auch Probleme, wenn man eine Rotationsdifferenz anteilig verteilen will. Wenn man von drei gegebenen Winkeln den äußeren (also den zuerst angewendeten) ändert, muss man die anderen beiden neu berechnen. Da dies nicht trivial ist und es bei bestimmten Winkeln zu weiteren Effekten kommt (z. B. Gimbal Lock), habe ich auf eine Implementierung verzichtet. Für kleine Winkelfehler kann man diese Probleme allerdings ignorieren und die Winkel einfach als linear unabhängig betrachten. Dazu habe ich den Schalter `-L1` in das SLAM-Programm eingebaut.

3.2.2 Quaternionen

Eine andere Möglichkeit Winkel darzustellen ist, diese als Einheitsquaternionen zu schreiben. Hier gibt es mehrere Möglichkeiten, einen Bruchteil des Winkels zu berechnen. Die einfachste ist die Normierung zu ignorieren und das Quaternion als einen Vektor im $\mathbb{R}^4$ zu betrachten. Nun kann man durch Vektoraddition und Skalierung näherungsweise die neuen Winkel berechnen. Allerdings haben die Ergebnisse nicht die Länge 1, sind also keine Rotationen, sodass man sie noch normieren muss. Besser ist es, die multiplikativen Eigenschaften der Quaternionen zu nutzen. Für eine initiale Rotation $\mathbf{q}_1$ und eine Fehlerkorrektur $\mathbf{q}_2$ ist $\mathbf{q}_3 = \mathbf{q}_2 * \mathbf{q}_1^{-1}$ die Kombination beider Rotationen. Wenn man $\mathbf{q}_3$ mit den berechneten Gewichten multipliziert, erhält man immer noch ein Quaternion, welches wieder normiert werden muss. Für kleine Rotationen ist der Fehler allerdings vernachlässigbar, was im SLAM-Programm mit den neuen Schaltern `-L2` beziehungsweise `-L3` getestet werden kann.

Tabelle 3.1: Die ELCH-Varianten und ihre Abkürzungen und Schalter im SLAM-Programm.

ELCH-Variante	**Abkürzung**	**Schalter**
Eulerwinkel	ELCH 1	-L1
Quaternionen	ELCH 2	-L2
Einheitsquaternionen	ELCH 3	-L3
SLERP	ELCH 4	-L4

3.2.3 SLERP

Im Gegensatz zu allen bisher vorgestellten Lösungsansätzen beachtet die Spherical Linear Interpolation (SLERP, von Shoemake, [27]) die Normiertheit der Quaternionen. Zwei Quaternionen spannen einen Kreisbogen auf, der in einer Ebene liegt. SLERP berechnet aus zwei Quaternionen und einem Faktor $w \in [0,1]$ ein Quaternion, welches auf diesem Kreisbogen liegt und den durch den Faktor gegebenen Abstand zu Anfang und Ende hat. SLERP braucht allerdings nur einen Faktor pro Quaternion und nicht für jede Komponente einen, wie es die Varianzberechnung in Abschnitt 2.3 liefert. Um den Algorithmus nicht unnötig kompliziert zu machen, addiere ich alle Rotationsvarianzen zu einem Faktor und berechne die ELCH-Korrektur nur für vier Komponenten. Im Programm habe ich für diese Korrektur den Schalter -L4 eingebaut.

Insgesamt gibt es die in Tabelle 3.1 aufgeführten ELCH-Varianten, welche im Abschnitt 4.1.2 verglichen werden.

3.2.4 Berechnung und Verteilung des Fehlers

In den theoretischen Überlegungen bin ich immer davon ausgegangen, dass der Anfang der Schleife im Ursprung liegt, praktisch wird dies aber so gut wie nie der Fall sein. Um den Fehler allgemein zu verteilen, muss man das Koordinatensystem entsprechend transformieren, was ich jetzt ausführen werde.

Im Folgenden definiert $\mathbf{P}_i^0$ die $\mathbb{R}^{4\times4}$-Transformationsmatrix des i-ten Scans im Koordinatensystem des 0-ten Scans und $\overline{\mathbf{P}}$ das Inverse von $\mathbf{P}$. Um zwischen den Koordinatensystemen der einzelnen Scans umzurechnen, ergibt sich folgende Formel:

$$\mathbf{P}_i^0 = \mathbf{P}_j^0 \mathbf{P}_i^j, \text{ bzw. } \mathbf{P}_i^j = \overline{\mathbf{P}_j^0} \mathbf{P}_i^0. \tag{3.1}$$

Die ICP-Verschiebung berechne ich nun zunächst im Koordinatensystem des Scans, der den Anfang der Schleife definiert (f-ter Scan). Damit wird das Ende der Schleife (der l-te Scan) auf den Ursprung gezogen und man erhält als Fehler:

$$\delta^f = \mathring{\mathbf{P}}_l^f \overline{\mathbf{P}_l^f},$$

$\mathring{\mathbf{P}}_l$ entspricht dabei dem l-ten Scan nach der ICP-Korrektur. Mit Formel 3.1 umgerechnet in das Koordinatensystem des 0-ten Scans folgt:

$$\delta^f = \overline{\mathbf{P}_f^0} \mathring{\mathbf{P}}_l^0 \overline{\mathbf{P}_f^0 \mathbf{P}_l^0}. \tag{3.2}$$

Die Verschiebung δ wird nun für jeden Scan angepasst (aus δ wird δ_i für den i-ten Scan, wie in den vorhergehenden Abschnitten beschrieben). Damit berechnen sich die neuen Posen für die Scans:

$$\hat{\mathbf{P}}_i^f = \delta_i^f \mathbf{P}_i^f .$$

Dabei wird auch der 0-te Scan transformiert (genauer: alle bis auf den f-ten). Um dies rückgängig zu machen, wird die inverse Transformation des 0-ten Scans auf alle anderen angewandt:

$$\tilde{\mathbf{P}}_i^f = \overline{\delta_0^f} \delta_i^f \mathbf{P}_i^f .$$

Wieder wie in Formel 3.1 umgerechnet in das ursprüngliche Koordinatensystem ergibt sich:

$$\tilde{\mathbf{P}}_i^0 = \mathbf{P}_f^0 \tilde{\mathbf{P}}_i^f .$$

Genauso auf der rechten Seite:

$$\tilde{\mathbf{P}}_i^0 = \mathbf{P}_f^0 \overline{\delta_0^f} \delta_i^f \overline{\mathbf{P}_f^0} \mathbf{P}_i^0 . \tag{3.3}$$

Für einzelne Scans lässt sich die Formel noch vereinfachen. Für den 0-ten Scan ergibt sich die Identität, wie man leicht nachprüft. Für den f-ten Scan ergibt sich (mit $\delta_f^f = \mathbf{I}$):

$$\tilde{\mathbf{P}}_f^0 = \mathbf{P}_f^0 \overline{\delta_0^f} .$$

Und für den l-ten Scan mit $\delta_l^f = \delta^f$ ergibt sich aus Formel 3.3 mit Formel 3.2:

$$\tilde{\mathbf{P}}_l^0 = \mathbf{P}_f^0 \overline{\delta_0^f} \overline{\mathbf{P}_f^0} \mathring{\mathbf{P}}_l^0 \overline{\overline{\mathbf{P}_f^0} \mathbf{P}_l^0} \overline{\mathbf{P}_f^0} \mathbf{P}_l^0 ,$$

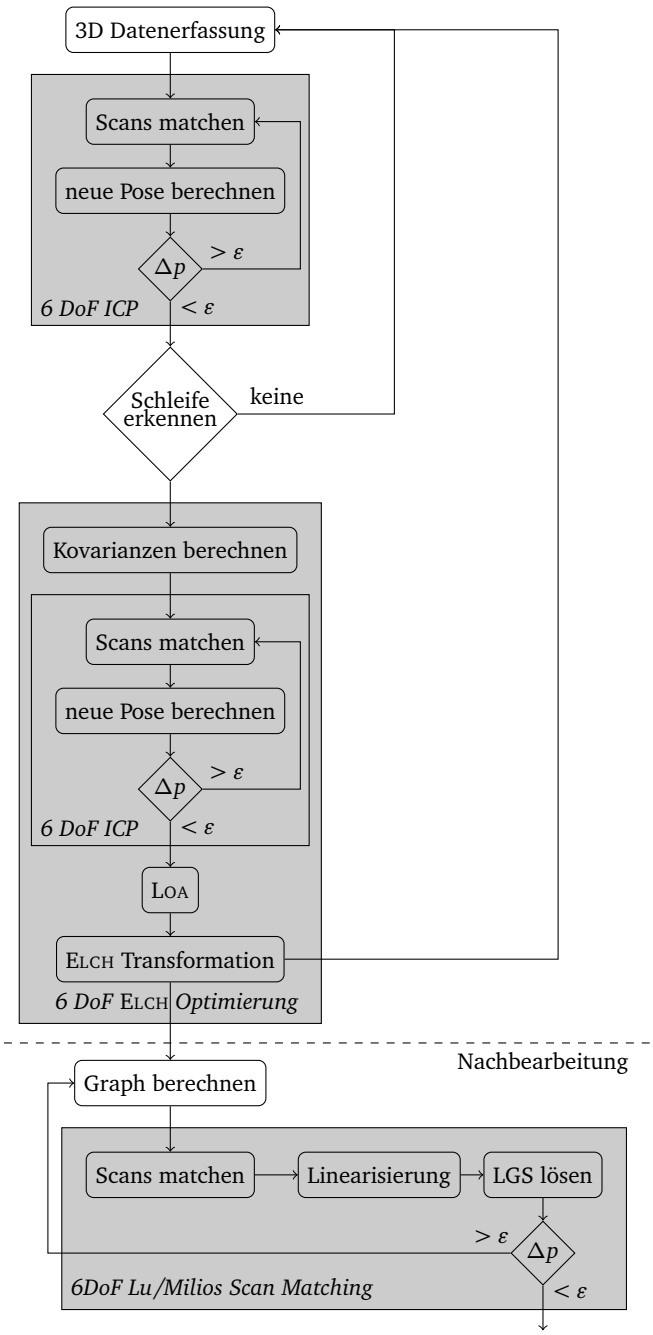
und gekürzt:

$$\tilde{\mathbf{P}}_l^0 = \mathbf{P}_f^0 \overline{\delta_0^f} \overline{\mathbf{P}_f^0} \mathring{\mathbf{P}}_l^0 .$$

3.3 Die LOA-Implementierung

Die ELCH-Strategie (Abbildung 3.7) gliedert sich in zwei Teile. Als erstes wird Anfang und Ende einer Schleife gesucht und auf diese ICP angewendet. Das daraus erhaltene δ muss nun über alle Laserscans verteilt werden. Dies erledigt der Loop Optimizer-Algorithmus (LOA), der für jeden Laserscan eine Gewichtung $w \in [0,1]$ ausrechnet. Multipliziert mit δ ergibt dies die Verschiebung für den Laserscan.

LOA (Algorithmus 1) bekommt als Übergabewerte den bisherigen Graphen G mit neu berechneten Varianzen als Kantengewichten und *ohne* Kante für die neu gefundene Schleife, den ersten Knoten (f) und den letzten Knoten (l) der neuen Schleife. Diesen weist er als Verankerung die Werte 0 und 1 zu und fügt sie in die Liste Ω der noch zu bearbeitenden Knoten ein. Jetzt sucht er mit dem Dijkstra-Algorithmus den kürzesten Weg zwischen zwei beliebigen Elementen in der Liste – im ersten Durchlauf sind dies f und l – und ordnet allen Knoten entlang dieses Weges ein Gewicht zwischen 0 und 1 zu. Trifft er auf dem Weg auf Knoten mit einem Grad größer zwei, also Knoten, die noch



$\mathcal{O}(N \log N)$ (I)

$\mathcal{O}(N)$ (II)

$\mathcal{O}(1)$ (III)

$\mathcal{O}(n)$ (IV)

$\mathcal{O}((l+n)N \log N)$ (V)

$\mathcal{O}(N \log N)$ (VI)

$\mathcal{O}(N)$ (VII)

$\mathcal{O}(1)$ (VIII)

$\mathcal{O}(ln \log n)$ (IX)

$\mathcal{O}(n)$ (X)

$\mathcal{O}(n^2)$ (XI)

$\mathcal{O}((l+n)N \log N)$, $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$ (XII)

$\mathcal{O}(1)$ (XIII)

Abbildung 3.7: Erweiterung von Abbildung 2.1 um den ELCH-Ansatz (V bis X). Die Nachbearbeitung mit LUM (XI bis XII) ist optional.

Algorithm 1 Loop Optimizer

Input: Graph $G = (V, E)$
first vertex v_f
last vertex v_l
edge costs $c_{l,k} : E \to \mathbb{R}_+$
Output: vertex weights $w_i : V \to [0, 1]$

$w_f \leftarrow 0$
$w_l \leftarrow 1$
$\Omega \leftarrow \{v_f, v_l\}$ /* Loop Closing */
/* Dijkstra returns a path $p := (v_s, v_1, v_2, \ldots, v_n, v_e)$ */
/* and minimal costs $d(v_s, v_s), d(v_s, v_1), \ldots, d(v_s, v_e)$ */
while find shortest path between any two vertices $\{v_s, v_e\} \in \Omega$ with Dijkstra **do**
  for all vertices v_i on the path p **do**
    $w_i \leftarrow w_s + \frac{d(v_s, v_i)}{d(v_s, v_e)}(w_e - w_s)$
    if $deg(v_i) > 2$ **then** /* i.e. loop */
      $\Omega = \Omega \cup \{v_i\}$
    end if
  end for
  remove edges of path p in G
  if $deg(v_s) = 0$ **then**
    $\Omega = \Omega \setminus \{v_s\}$
  end if
  if $deg(v_e) = 0$ **then**
    $\Omega = \Omega \setminus \{v_e\}$
  end if
end while
while $\Omega \neq \emptyset$ **do** /* Error Propagation */
  select $v_i \in \Omega$
  for all neighbors v_n of v_i **do**
    $w_n \leftarrow w_i$
    delete edge $\{v_i, v_n\}$
    if $deg(v_n) > 0$ **then**
      $\Omega = \Omega \cup \{v_n\}$
    end if
  end for
  $\Omega = \Omega \setminus \{v_i\}$
end while

zusätzliche Kanten zum gefundenen kürzesten Weg haben, werden diese an die Liste Ω angehängt und in den folgenden Durchläufen abgearbeitet. Nachdem der gefundene Weg im Graphen entfernt wurde, werden Start- und Endknoten in der Liste gelöscht, wenn keine weiteren Wege von ihnen ausgehen. Dieses Vorgehen wird wiederholt, solange noch weitere Schleifen in Ω vorhanden sind.

Am Ende verbleiben in der Liste nur Knoten, von denen kein Weg zurück auf ein Element der Liste führt (Abzweigungen). Diese werden nun einzeln abgearbeitet und allen Nachbarknoten wird das selbe Gewicht wie dem Ursprungsknoten zugeordnet.

In Abschnitt 3.1.5 hatte ich ein Beispiel gezeigt, bei dem es mehrere Möglichkeiten gibt, die Kantengewichte zu berechnen. Da die LOA-Implementierung eine Liste für die Menge Ω nutzt, ist immer eine eindeutige Reihenfolge für die zu bearbeitenden Knoten gegeben. Dies bedeutet auch, dass bei mehreren Möglichkeiten immer die genommen wird, die am nächsten am Hauptkreis liegt, da deren Knoten als erstes in die Liste eingefügt werden. Außerdem stellt LOA sicher, dass die berechneten Gewichte lokal konsistent sind. Ein Knoten wird also immer ein größeres oder gleiches Gewicht als seine Vorgänger und ein kleineres oder gleiches Gewicht als seine Nachfolger zugewiesen bekommen. Bei einem Knoten mit zwei Kanten sind Vorgänger und Nachfolger dabei die vorherige bzw. nachfolgende Pose in der Trajektorie. Bei Knoten mit mehreren Kanten wird die Reihenfolge durch die Schleifenlänge bestimmt (siehe Abschnitt 3.1.4).

3.4 Laufzeitvergleich

In Abbildung 3.7 stehen auf der rechten Seite die Laufzeiten der einzelnen Algorithmen in $\mathcal{O}$-Notation. Für die Experimente in Kapitel 4 liegt die Anzahl der Datenpunkte in einem Scan bei ungefähr $N \approx 16\,000$. Die Anzahl der Laserscans ist in der Größenordnung von $n \approx 1\,000$ und die Anzahl der Schleifen ist sehr viel kleiner $l \approx 10$. Verglichen mit Abbildung 2.1 wird die Gesamtlaufzeit nun nicht mehr von LUM dominiert, sondern von ICP. Im ELCH-Teil dominiert die Berechnung der Kovarianzen, welche aber im Gegensatz zu LUM nicht iterativ durchgeführt wird und deswegen nicht so viel Zeit beansprucht. Wie stark sich diese Laufzeitverkürzung im konkreten Fall auswirkt, ist in Abschnitt 4.1.3 ausgeführt.

3.5 Mehrere Roboter

Der ELCH-Ansatz kann auch genutzt werden, um die Karten mehrerer Roboter zu fusionieren, die von verschiedenen Startpositionen losgefahren sind. Dazu muss eine initiale Poseschätzung aller Roboter in einem globalen Koordinatensystem vorhanden sein. Wenn ein Roboter die Trajektorie eines anderen das erste Mal kreuzt, wird ein Loop Closing Event ausgelöst (Abbildung 3.8 links). Allerdings ist noch keine Schleife vorhanden und der Dijkstra-Algorithmus in der LOA-Implementierung bricht, ohne eine Schleife gefunden zu haben, ab. Jeder Knoten auf der Trajektorie des ersten Roboters (A) erhält damit das Gewicht 0, während die des zweiten (B) ein Gewicht von 1 bekommen. So wird die Trajektorie des zweiten Roboters bezüglich der Poseschätzung des ersten korrigiert (Abbildung 3.8 rechts). Alle weiteren Schleifen in den Trajektorien sind nun echte Schleifen und können wie bisher behandelt werden. Diese Strategie kann man auch nutzen, um Karten des selben Gebiets, die mit nur einem Roboter hintereinander aufgenommen wurden, zu kombinieren (siehe Abschnitt 4.3).

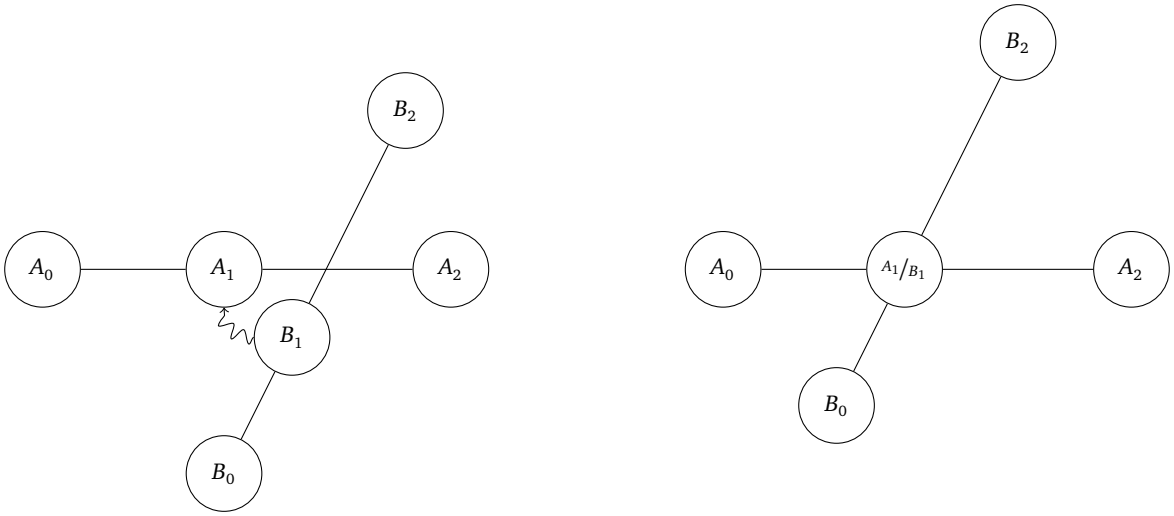


Abbildung 3.8: Anwendung der ELCH-Strategie auf die Trajektorie von zwei Robotern. Der zweite Roboter (*B*) kreuzt dabei die Trajektorie des ersten (*A*) und alle seine Posen werden entsprechend korrigiert.

4 Experimente

Die zuvor entwickelten ELCH-Strategien werden nun auf zwei Datensätze angewendet, die beide auf dem Gelände der Universität Hannover aufgenommen wurden (Abbildung 4.1). Videos von den Experimenten können unter [29] angeschaut werden.

Abbildung 4.1: Luftbild der Universität Hannover (Quelle: Google Maps).

4.1 Der große Datensatz

Der Datensatz HANNOVER 2 (von [20], siehe Abbildung 4.2) wurde von Oliver Wulf auf dem Gelände der Uni Hannover aufgenommen und enthält 6 Schleifen. Die Odometriedaten enthalten keine großen Sprünge und die Scans liegen recht dicht beieinander, sodass der ICP-Algorithmus relativ schnell und gut konvergiert. Auch der akkumulierte Fehler innerhalb der Schleifen ist nicht sehr groß, weshalb alle in Abschnitt 3.2 beschriebenen Methoden brauchbare Ergebnisse liefern. Nutzt man die einfache Quaternionen-Implementierung (`-L2`) aus Abschnitt 3.2.2, so tritt ein interessanter Effekt auf. Zwischen Markierung *F* und *E* in Abbildung 4.2 fährt der Roboter eine längere Strecke den selben Weg wie zuvor. Am Anfang dieses Stücks (Markierung *a* in Abbildung 4.3) wird zunächst eine Schleife geschlossen und die Posen werden korrigiert. Während der weiteren Fahrt divergiert aber die neue Pose soweit von den alten, dass ein neues Loop Closing Event ausgelöst wird (Markierung *b* in Abbildung 4.3). Da die beiden Strecken aber fast gleich sind, werden die Posen aufeinander gezogen und bilden sofort wieder den Anfang für eine neue Schleife. Dies kann prinzipiell bei jeder ELCH-Variante passieren und hängt von der Güte der Rohdaten ab. Die dabei zusätzlich eingefügte Kante im SLAM-Graph ist aber nicht negativ zu sehen, da sie eine feiner abgestufte Verteilung des Fehlers ermöglicht.

Wenn man in Abbildung 4.3 die blaue Trajektorie und die roten Korrekturen zusammen nimmt, erhält man den kompletten SLAM-Graphen, welcher von den entsprechenden ELCH-Varianten erstellt wird.

4.1.1 Der globale Fehler

Um die Qualität der mit der ELCH-Strategie gewonnenen Trajektorie zu bewerten, habe ich sie mit einem Referenzdatensatz verglichen. Dazu wurden die Daten aus einem Überflug mit einem Laserscanner (Abbildung 4.4 oben links) mit denen des Katasteramtes fusioniert (Abbildung 4.4 oben rechts) und die einzelnen Scans dagegen gematcht (Abbildung 4.4 unten). In Abbildung 4.5 wird die ELCH- und LUM-Trajektorie mit der so gewonnenen Referenztrajektorie verglichen, um die abschließende LUM-Korrektur zu verdeutlichen. Man kann erkennen, dass der Fehler um so größer wird, je weiter weg die Scanpose vom Ursprung liegt und je weniger Schleifen sie mit der restlichen Trajektorie verbinden. Dies liegt daran, dass der Anfang der Schleife von der initialen ICP-Korrektur und das Ende von ELCH recht genau platziert werden. In der Mitte der Schleife bleibt der Fehler dabei am größten, weshalb die Scans nicht mehr so gut platziert werden können.

In Abbildung 4.6 ist der Translationsfehler als euklidischer Abstand zum wahren Wert dargestellt. Verglichen mit den vorherigen Abbildungen scheint der Fehler recht groß zu sein, allerdings entfällt ein Großteil davon auf die Höhenkoordinate, was aber am Datensatz zu liegen scheint, da es unabhängig vom gewählten Algorithmus auftaucht. Zusätzlich habe ich testweise auch einen Durchlauf mit vertauschten Koordinaten durchgeführt, um Fehler in der Implementierung auszuschließen.

Den Rotationsfehler anschaulich darzustellen, ist nicht einfach. Nimmt man z. B. die Eulerwinkel, so müsste man drei Winkel darstellen, die sich auch noch gegenseitig beeinflussen. Deswegen rechne ich beide Rotationen, die korrekte und die mit SLAM gefundene, zunächst in Quaternionen um und bilde dann das innere Produkt. Da die Quaternionen die Länge 1 haben, ergibt sich so der Winkel zwischen ihnen (vergleiche Kuffner [15]). Mit $\mathbf{x}_i$ als Translationsvektor und $\mathbf{q}_i$ als

Abbildung 4.2: Oben: Draufsicht auf die mit ELCH erstellte Karte aus dem Datensatz HANNOVER 2. Die Scans sind entlang der roten Trajektorie in folgender Reihenfolge aufgenommen worden: *A-B-C-D-A-B-E-F-A-D-G-H-I-J-H-K-F-E-L-I-K-A*. Scans, welche in der Arbeit besonders ausgewertet werden, sind gelb markiert. Unten: Schrägsicht aus Richtung Westen.

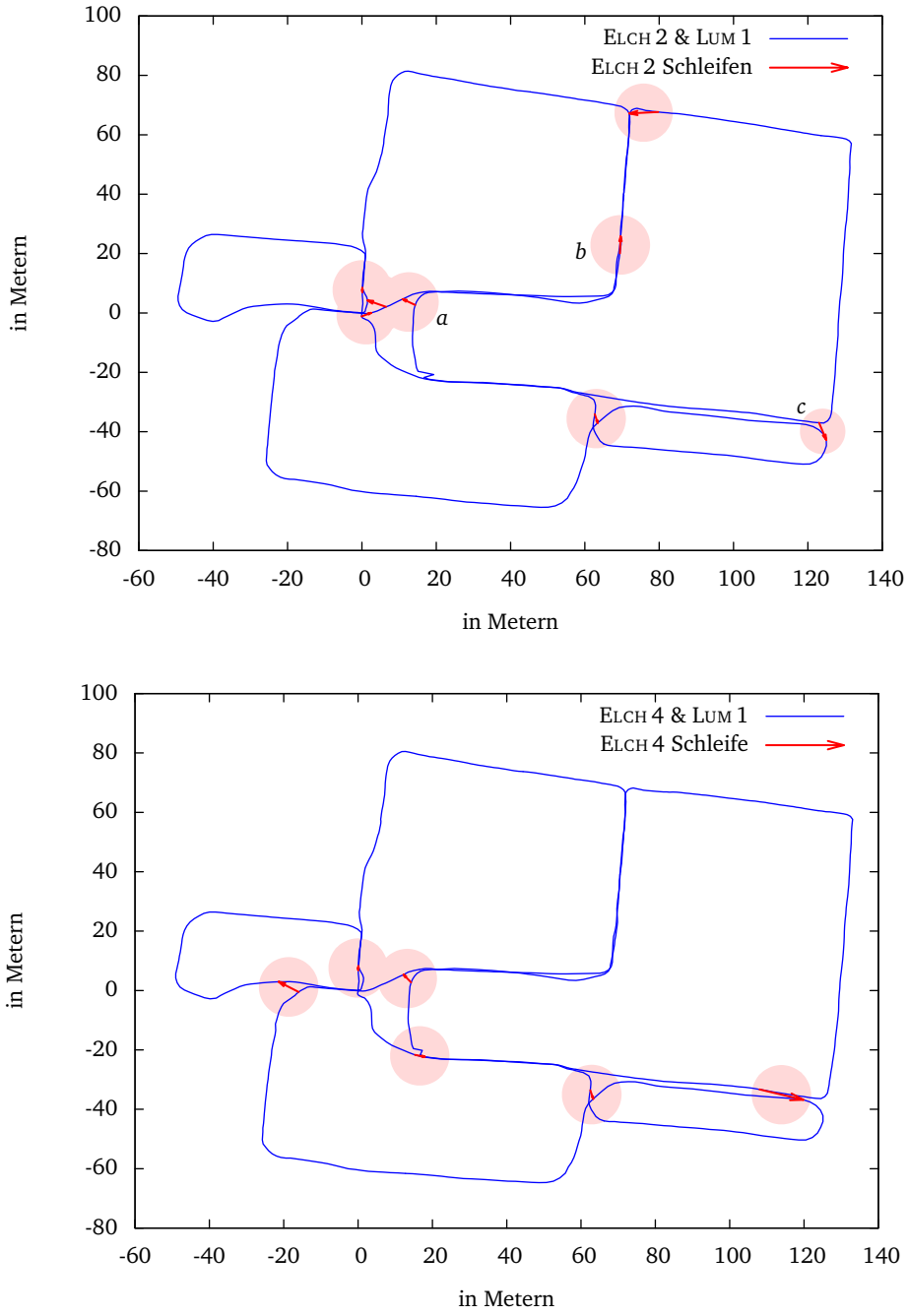


Abbildung 4.3: Oben: SLAM-Graph von ELCH 2 bestehend aus der Trajektorie (blau) und den durch Schleifen hinzugefügten Kanten (rote Pfeile, rosa hinterlegt). Bei Markierung *a* fängt der Roboter an, die selbe Strecke noch einmal zu fahren. Dabei divergieren beide Trajektorien so weit, dass bei *b* ein neuer Loop Closing Event ausgelöst wird. Im Gegensatz zu Wulf et al. [33] findet ELCH auch die letzte große Schleife im Datensatz und kann sie korrigieren (*c*). Unten: SLAM-Graph von ELCH 4. Durch die bessere Verteilung des Rotationsfehlers treten die im oberen Graphen gezeigten Effekte nicht mehr auf und der Graph enthält nur noch die wirklich im Datensatz vorhandenen Schleifen.

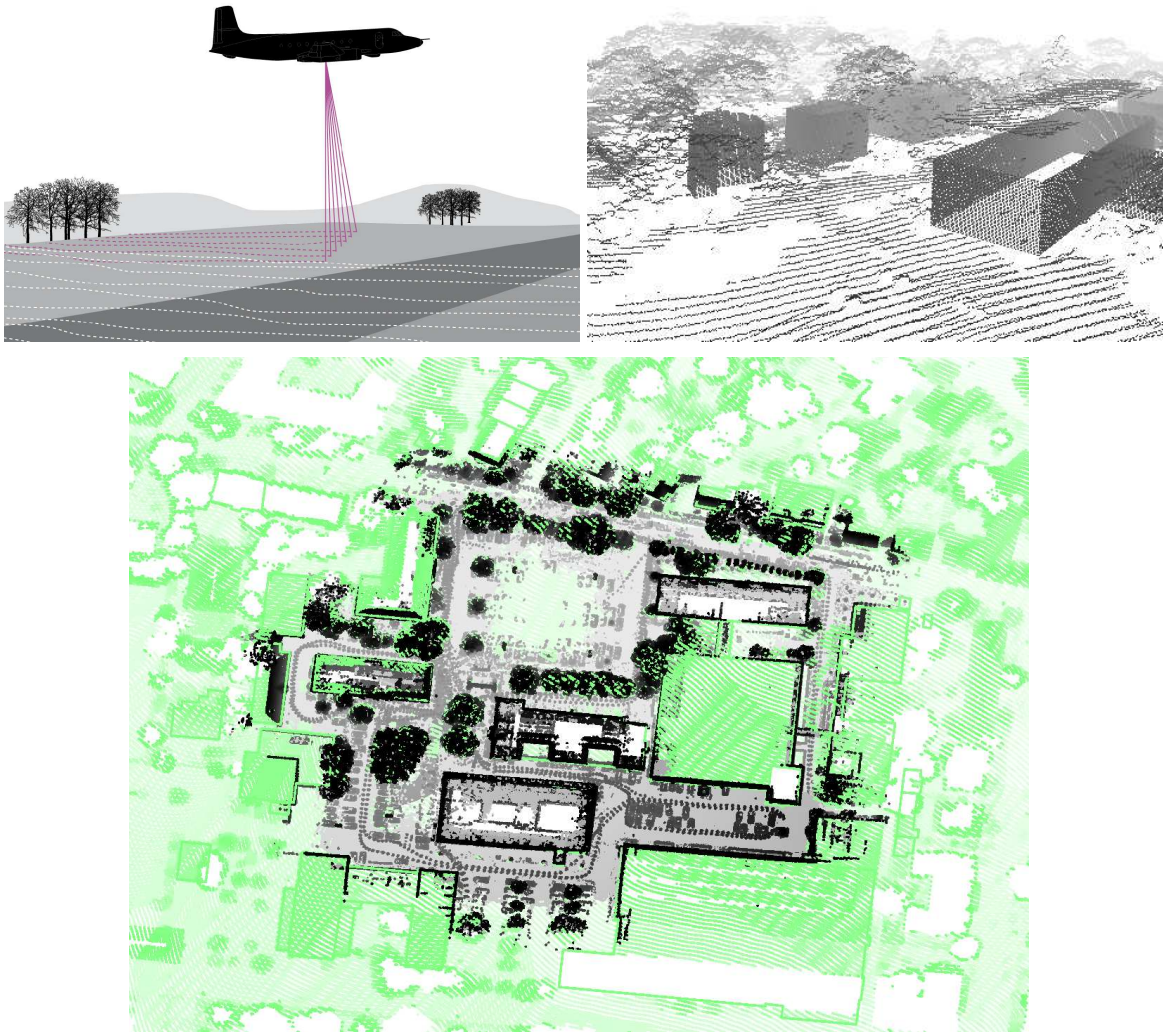


Abbildung 4.4: Oben links: Laserabtastung mit einem Flugzeug. Oben rechts: 3D Szene des Referenzdatensatzes bestehend aus den Daten vom Katasteramt (rechtwinklige Gebäudemauern) und denen des Überflugs. Unten: Gegen den Referenzdatensatz (grün) gematchte Scans des Datensatzes HANNOVER 2 (schwarz).

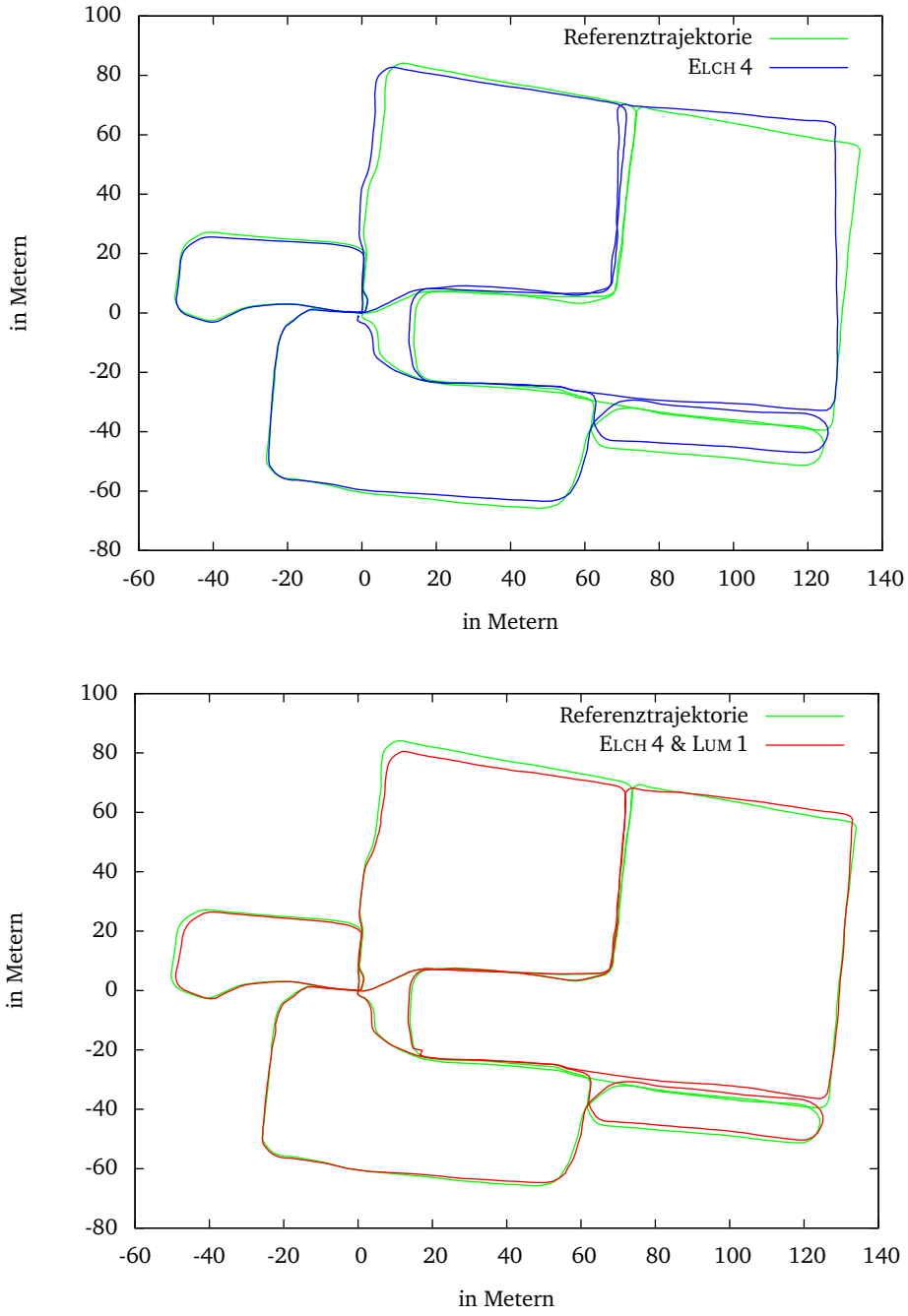


Abbildung 4.5: Oben: Vergleich der nur mit ELCH 4 errechneten Trajektorie (blau) von HANNOVER 2 mit der Referenztrajektorie (grün). Man sieht, dass der Fehler in der Mitte der Schleife und am weitesten weg von Ursprung am größten ist, z. B. oben rechts, da ELCH dort am wenigsten Informationen hat, um die Pose zu korrigieren. Unten: Die von ELCH 4 und LUM 1 errechnete Trajektorie (rot) von HANNOVER 2. Verglichen mit oberen Graphen erkennt man, dass LUM hauptsächlich den Mittelpunkt großer Schleifen (z. B. oben rechts) verschiebt und das Ergebnis noch besser zur Referenztrajektorie (grün) passt.

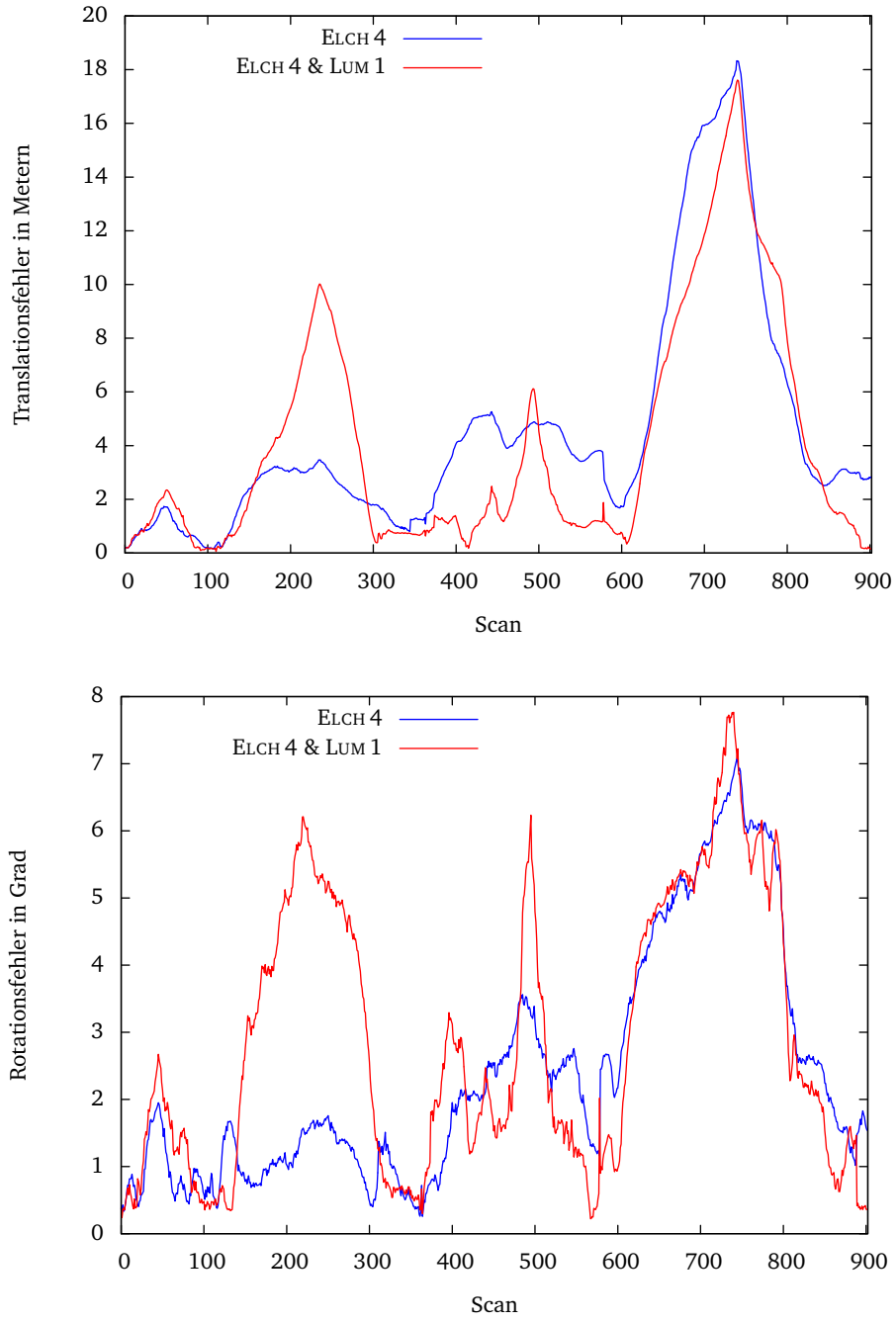


Abbildung 4.6: Oben: Translationsfehler jedes Scans aus dem Datensatz HANNOVER 2 von ELCH 4 (blau) und ELCH 4 mit LUM 1 (rot). Ein Großteil des Fehlers liegt in der Höhenkoordinate (vergleiche Abbildung 4.5), worauf in Abschnitt 4.1.1 eingegangen wird. Unten: Rotationsfehler der Scans aus dem Datensatz HANNOVER 2.

Tabelle 4.1: Mittelwerte und Standardabweichungen der Scans.

Algorithmus	**Translation [m]**	**Translation [m] (nur x und y)**	**Rotation [°]**
ICP	9,16 ± 6,70	8,35 ± 6,22	3,31 ± 2,64
ICP (mit Metascan)	2,77 ± 2,24	1,96 ± 2,21	1,35 ± 0,72
LUM 1	6,27 ± 6,96	3,88 ± 3,98	3,50 ± 2,67
ELCH 1	11,54 ± 16,09	7,29 ± 9,06	10,7 ± 8,28
ELCH 2	7,91 ± 8,38	5,45 ± 5,31	3,08 ± 2,29
ELCH 3	5,08 ± 5,18	3,62 ± 2,98	2,19 ± 1,27
ELCH 4	4,53 ± 4,38	2,73 ± 2,18	2,40 ± 2,77
ELCH 4 & LUM 1	4,05 ± 4,29	1,38 ± 1,00	2,90 ± 2,31

Rotationsquaternion des i-ten Scans ergeben sich insgesamt folgende Formeln zur Berechnung des Fehlers:

$$e_{\text{translation}} = \left\| \mathbf{x}_i - \mathbf{x}_{i,\text{ref}} \right\|,$$
$$e_{\text{rotation}} = \arccos \left| \mathbf{q}_i \cdot \mathbf{q}_{i,\text{ref}} \right|.$$

Wenn man die Graphen für den Translations- und Rotationsfehler vergleicht, sieht man, dass sie einen ähnlichen Kurvenverlauf haben. Daran kann man erkennen, dass die Scans lokal korrekt ausgerichtet sind. Würden beide Kurven an einer Stelle sehr voneinander abweichen, z. B. ein kleiner Translations- und ein sehr großer Rotationsfehler, so würde das darauf hindeuten, dass der Scan nicht richtig gematcht wurde, also lokal nicht konsistent ist.

Um die in Abschnitt 3.2 beschriebenen ELCH-Varianten global zu bewerten, habe ich die Mittelwerte und Standardabweichungen der Posefehler aller Scans berechnet (Tabelle 4.1). Da der ICP-Algorithmus ohne Metascan, welcher LUM und den ELCH-Algorithmen vorgeschaltet ist, teilweise recht große Fehler in der Höhe (z-Koordinate) macht, habe ich zusätzlich die Werte nur für x und y berechnet. Diese sind auch besser mit den Trajektorien in Abbildung 4.5 vergleichbar. Nimmt man nur diese Werte, so könnte man annehmen, dass ICP mit Metascan die besten Ergebnisse liefert. Dies ist aber nur scheinbar so, da ICP die Scans nur einmal registriert und einzelne dabei falsch positioniert werden können, wodurch das Resultat nicht global konsistent ist. Dieser Fehler wird allerdings in den Werten herausgemittelt. Außerdem braucht der ICP-Algorithmus mit Metascan sehr viel länger, um das Ergebnis zu berechnen, worauf ich in Abschnitt 4.1.3 eingehe.

Vergleicht man die einzelnen ELCH-Varianten, so sieht man deutlich, dass ELCH 4 am genauesten ist. Im nächsten Abschnitt werde ich dieses Ergebnis anhand einzelner Scans verdeutlichen. Die abschließende LUM-Korrektur von ELCH 4 ergibt global konsistente Scanposen, weshalb der Mittelwert der Rotation wieder etwas größer wird.

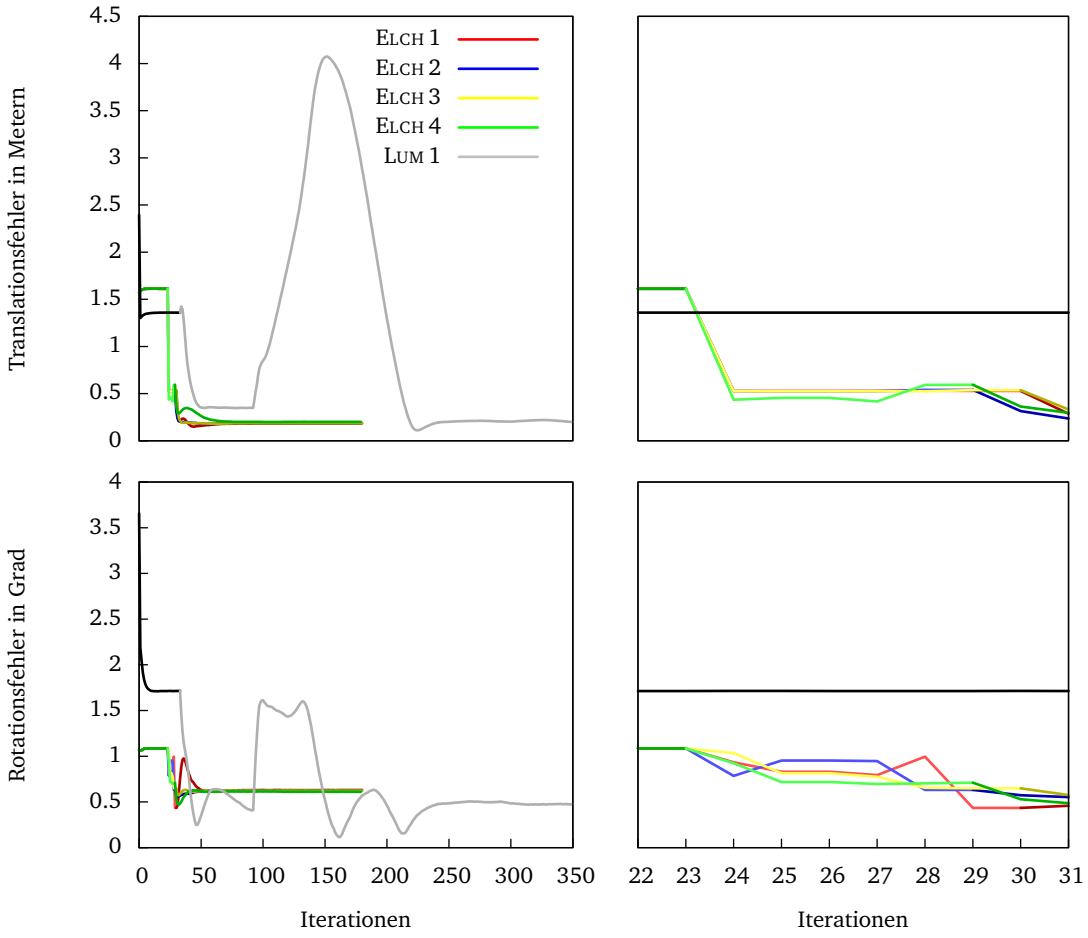


Abbildung 4.7: Konvergenzgraph des 108-ten Laserscans aus dem Datensatz HANNOVER 2 für alle in der Arbeit beschriebenen Algorithmen. Auf der rechten Seite ist ein Ausschnitt der linken zu sehen, bei dem die ELCH-Korrekturen hervorgehoben sind (hellere Farbe).

4.1.2 Konvergenz einzelner Scans

Für drei ausgewählte Scans, welche in Abbildung 4.2 gelb markiert sind, habe ich außerdem den Fehler während des kompletten Programmdurchlaufs geplottet. Auf der linken Seite der Abbildungen 4.7 bis 4.9 ist der Translations- und Rotationsfehler aller vier ELCH-Varianten und der LUM-Version mit Eulerwinkeln zum Vergleich geplottet. Auf der rechten Seite sieht man einen Ausschnitt der linken Seite, in dem die ELCH-Korrekturen, durch einen helleren Farbwert abgesetzt, vergrößert dargestellt sind. Jeder Scan startet in der 0-ten Iteration mit der initialen Roboterpose, welche dann anhand des vorhergehenden, schon gematchten Scans korrigiert wird. Dies bewirkt die scheinbar unterschiedlichen Anfangsfehler. Danach wird der Scan zunächst mit ICP gegen den vorhergehenden Scan gematcht (dunkle Linie, ungefähr 40 Iterationen). In der LUM-Version folgt nun direkt die globale Relaxation (grau). In den ELCH-Varianten folgt am Ende noch ein LUM-Aufruf, welcher wieder mit dunkleren Farben dargestellt ist.

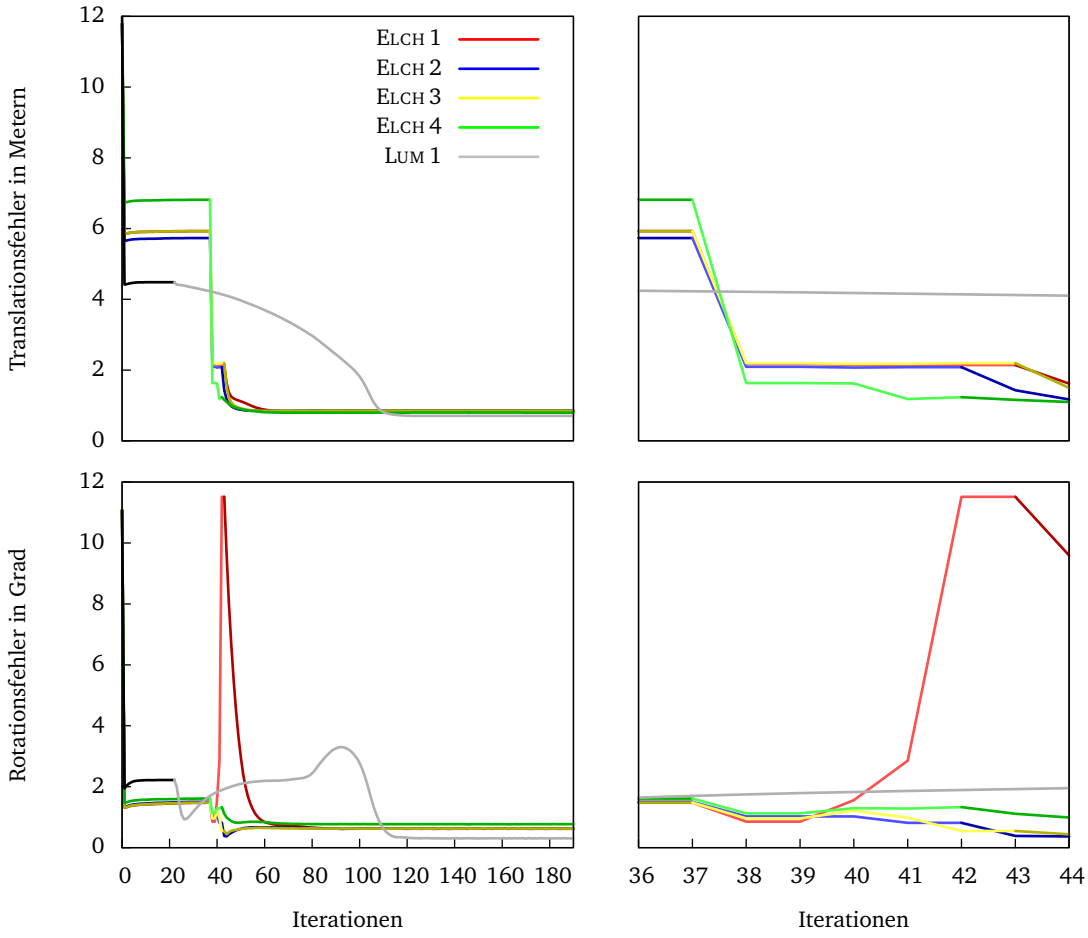


Abbildung 4.8: Konvergenzgraph des 344-ten Laserscans (vergleiche Abbildung 4.7).

Der 108-te Scan (Abbildung 4.7) des Datensatzes liegt kurz vor Ende der ersten Schleife (Markierung *D* in Abbildung 4.2) und damit recht nahe am Ursprung der Trajektorie. Der erste ELCH-Aufruf verbindet ihn im SLAM-Graph mit einem Scan nahe des Ursprungs und die berechnete neue Pose hat nur noch einen relativ kleinen Fehler. Alle weiteren Schleifenschlüsse verschieben den Scan deshalb nur noch unwesentlich. Mit LUM kann es dagegen passieren, dass er bei erneuten Durchläufen weit durch die Gegend geschoben wird, ehe er zu seiner endgültigen Position konvergiert, obwohl die Pose nach dem ersten Durchlauf (bis Iteration 90) schon recht gut war. Zwischen den einzelnen ELCH-Varianten ist kein großer Unterschied zu sehen, da der Rotationsfehler des Scans minimal ist.

Abbildung 4.8 zeigt den 344-ten Scan (Markierung *A* in Abbildung 4.2), welcher sich fast am Ende der zweiten Schleife befindet. Im Gegensatz zur ersten Schleife braucht LUM hier deutlich länger, bis er konvergiert. Außerdem sieht man deutlich, dass der Scan auf Grund der Schleife im SLAM-Graph erst langsam verschoben wird und erst am Ende eine Kante zum Anfang der Schleife eingefügt wird, welche die Konvergenz beschleunigt. Bei den ELCH-Varianten kann man den Einfluss der Rotation erkennen und dass die naive Berechnung über Eulerwinkel (ELCH 1) zu großen Fehlern führt.

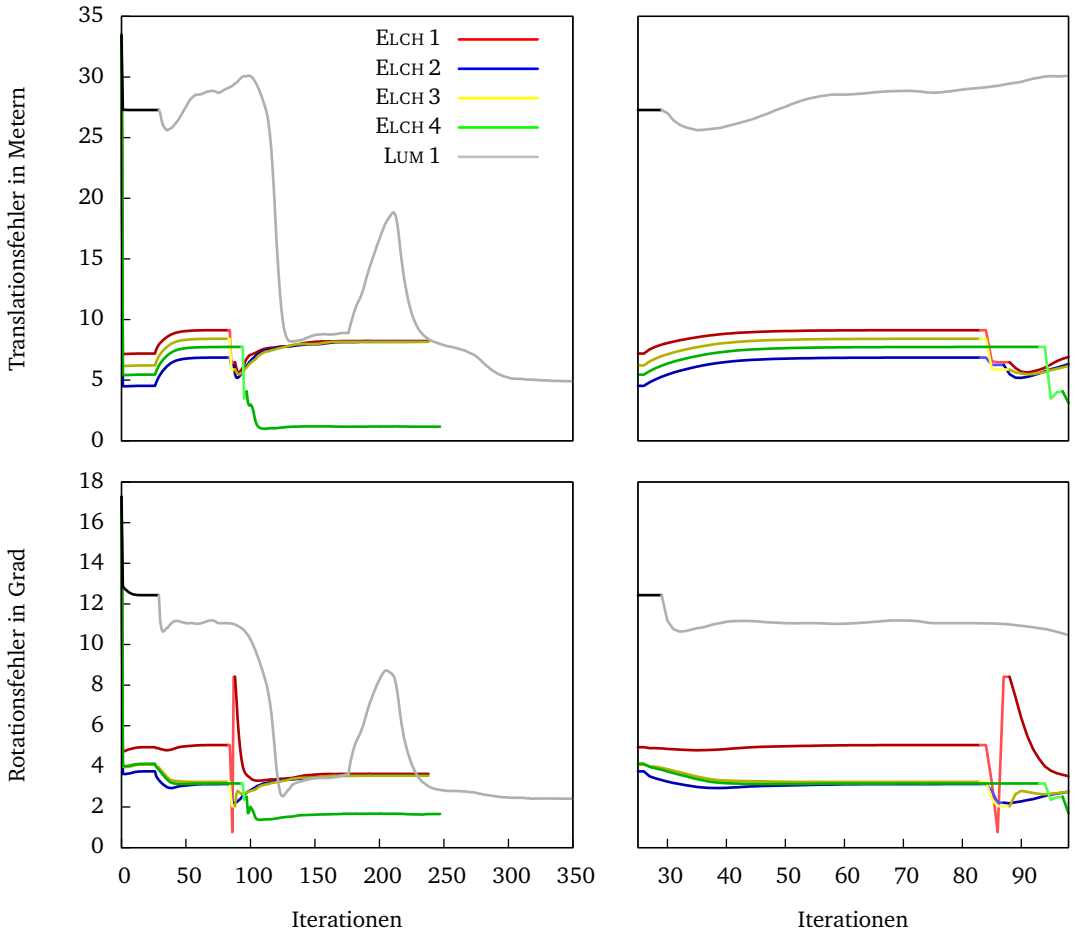


Abbildung 4.9: Konvergenzgraph des 556-ten Laserscans (vergleiche Abbildung 4.7).

Als letztes Beispiel habe ich in Abbildung 4.9 den 556-ten Scan gewählt, welcher in allen ELCH-Varianten eine Schleife weit weg vom Ursprung schließt (Markierung *H* in Abbildung 4.2). Hierbei sind auf der rechten Seite zusätzlich die ICP-Iterationen zum Schließen der Schleife eingezeichnet, weswegen der Scan erst wieder schlechter positioniert wird. Erst in den folgenden ELCH-Aufrufen (hellere Farbe) wird die Schleife mit weiteren verbunden und damit der Scan näher an seine richtige Pose gerückt. Dabei ist zu beachten, dass der erste ELCH-Aufruf den Scan zwar global gesehen schlechter positioniert, dieses aber vom Anfang der Schleife gesehen einen kleineren Fehler darstellt. Außerdem sieht man, dass die ELCH 4-Variante den Scan mit weiteren Schleifenschlüssen deutlich besser platziert als alle anderen Algorithmen.

4.1.3 Laufzeit und Gesamtbewertung

Insgesamt ergibt sich durch den Einsatz von ELCH sowohl ein Qualitäts- als auch ein Zeitgewinn. Dadurch, dass die Scans mit ELCH viel weniger durch die Gegend geschoben werden, sind die Ergebnisse nicht nur, wie oben dargestellt, besser, sondern es werden auch weitere Fehler korrigiert,

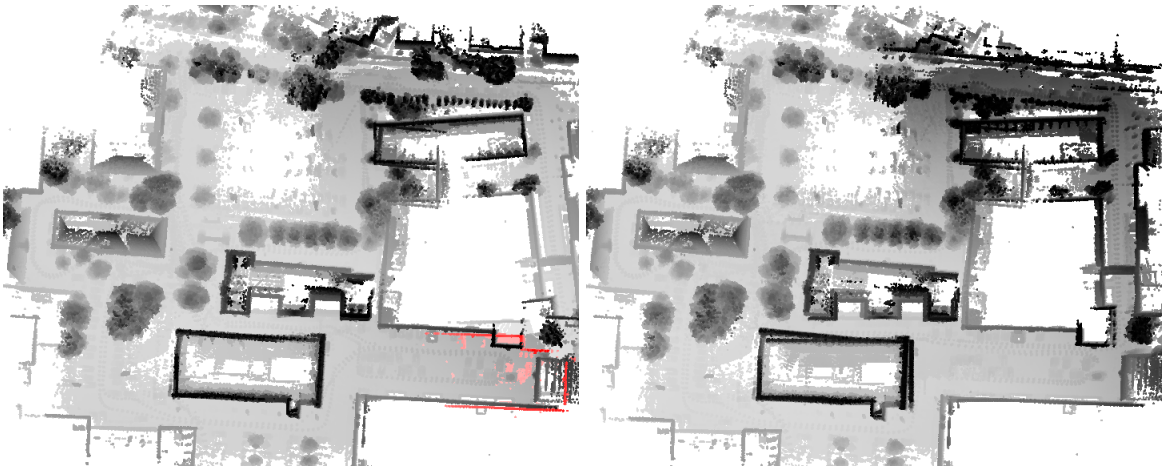


Abbildung 4.10: Schließen der in Abbildung 4.3 mit *c* markierten Schleife rechts unten. Links: das Ende der Schleife (rot) wird gegen den Anfang gematcht. Dabei werden vor allem die Höhe korrigiert, erkennbar an den Grauwerten. Rechts: Korrigierte Schleife.

Tabelle 4.2: Laufzeitvergleich auf einem Quadcore Intel Core 2 mit 2.66 GHz und 4 GB RAM für die komplette Verarbeitung des Datensatz HANNOVER 2.

Algorithmus	**Laufzeit [sec]**
ICP	49
ICP (mit Metascan)	3 222
LUM 1	4 831
LUM 2	3 299
ELCH	83
ELCH & LUM 1	384
ELCH & LUM 2	300

die LUM nicht findet. So schließt ELCH die in Abbildung 4.3 mit *c* markierte letzte große Schleife und kann damit, im Gegensatz zu LUM, alle Schleifen im Datensatz korrigieren (sie Abbildung 4.10). Zusätzlich ergibt sich ein sehr großer Laufzeitunterschied, wie in Tabelle 4.2 dargestellt. Dabei ist zu beachten, dass bei LUM alleine jedes Mal ein Aufruf erfolgt, wenn eine Schleife geschlossen wird. Die ELCH-Aufrufe fallen dagegen während des normalen Matchings kaum ins Gewicht, weshalb ich in der Tabelle auch nicht zwischen den einzelnen Varianten unterscheide. Außerdem entfällt ein Großteil der Zeit bei ELCH auf die Berechnung der Kovarianzen, was in Zukunft noch optimiert werden könnte (sie auch Ausblick in Kapitel 5).

4.2 Der kleinere Datensatz

Der Datensatz HANNOVER 1 [20] in Abbildung 4.11 ist ebenfalls von Oliver Wulf auf dem Gelände der Universität Hannover aufgenommen worden und zeigt zum Teil die selben Gebäude. Die Qualität der Rohdaten ist nicht ganz so gut wie in HANNOVER 2, sodass ICP Probleme bekommt, die Scans zu matchen.

Dies führt dazu, dass der 209-te Scan (ungefähr bei *B* in Abbildung 4.11) um ca. 18 Grad rotiert ist (Abbildung 4.12 links). ELCH findet den Fehler und gleicht ihn aus (Markierung *a* in Abbildung 4.13). Das Besondere ist, dass sich der Hauptteil des Fehlers auf genau einen Scan konzentriert und sich nicht, entgegen der eigentlichen Annahmen von ELCH, über alle Scans verteilt. Dabei erhalten alle Scans vor dem 209-ten eine größere Fehlerkorrektur als sie eigentlich bräuchten. Wie man in Abbildung 4.12 sieht, wird dies aber bei einem weiteren Schleifenschluss (Markierung *b* in Abbildung 4.13) behoben. Der Fehler wird dabei sogar so gut verteilt, dass das Gebäude unten rechts rechtwinklig wird (grüner Kreis). Dabei wird an der Stelle selber keine Schleife geschlossen, da die Scans vorher zu weit auseinander liegen. Im roten Kreis kann man außerdem die Auswirkungen des LOA-Algorithmus erkennen. Vor der Korrektur ist die markierte Struktur (auf der Seite liegendes I, siehe auch Abbildung 1.1) dreimal zu erkennen, während sie im rechten Bild nur noch einmal vorhanden ist.

Für drei, um den 209-ten liegende, Scans habe ich außerdem Konvergenzgraphen erstellt (vergleiche Abschnitt 4.1.2), die den Fehler über alle Iterationen darstellen. Der 200-te Scan in Abbildung 4.14 liegt dabei recht dicht vor dem 209-ten und man kann erkennen, dass vor allem seine Rotation nach dem ersten Schleifenschluss verschlechtert wird. Durch das Schließen der letzten Schleife wird der Fehler aber fast komplett kompensiert. Für den 245-ten Scan (Abbildung 4.15) ergibt sich nur eine deutliche Verbesserung, da er ungefähr auf der Hälfte zwischen dem 209-ten Scan und dem Schleifenende liegt. Der 308-te Scans (Abbildung 4.16) bildet das Ende der Schleife. Wie man erkennen kann, braucht der ICP-Algorithmus sehr viele Iterationen um den Fehler zu berechnen, weshalb man die weiteren ELCH-Korrekturen im Graphen fast gar nicht mehr sieht.

Bei Markierung *c* in Abbildung 4.13 kann man erkennen, dass zweimal fast die gleiche Schleife geschlossen wird. Dies liegt daran, dass an der Stelle eigentlich zwei Schleifen vorhanden sind. Zum einen die gefundene, zum anderen eine im kleinen Abstecher in y-Richtung. Da der Algorithmus zum Erkennen von Anfang und Ende einer Schleife aber sehr einfach ist, werden beide Male fast die selben Scans als Anfang und Ende genommen. Dies könnte man durch eine bessere Wahl der Parameter für die Schleifenerkennung korrigieren, insgesamt macht dies aber keine Probleme, da der Fehler während der kurzen Fahrt sehr klein bleibt. Die abschließende LUM-Korrektur verschiebt vor allem die im SLAM-Graphen wenig verbundenen Schleifen, wie man im Abbildung 4.17 sehen kann. Dabei wird auch der rechte Winkel aus Abbildung 4.12 wieder zerstört.

4.2.1 Wie man eine Referenztrajektorie generiert

Das Berechnen einer guten Referenztrajektorie hängt sehr von der Qualität der Rohdaten ab. Im Gegensatz zu Abschnitt 4.1.1 konnte ich für den kleineren Datensatz nicht auf eine fertige Referenztrajektorie zurückgreifen. Da beide Datensätze aber von der selben Umgebung gemacht wurden, sollte es auch möglich sein können, auf die gleiche Art eine Referenztrajektorie zu erstellen.

Abbildung 4.11: Oben: Draufsicht auf die mit ELCH 4 und LUM 1 erstellte Karte des Datensatzes HANNOVER 1. Der Roboter ist die Trajektorie in der Reihenfolge der Markierungen *A-B-C-D-A-E-F-A-B-G-C-B-A-H-I-A* abgefahren. Von den gelb eingefärbten Scans habe ich wieder Konvergenzgraphen erstellt. Unten: Schrägsicht aus Richtung Nordosten.

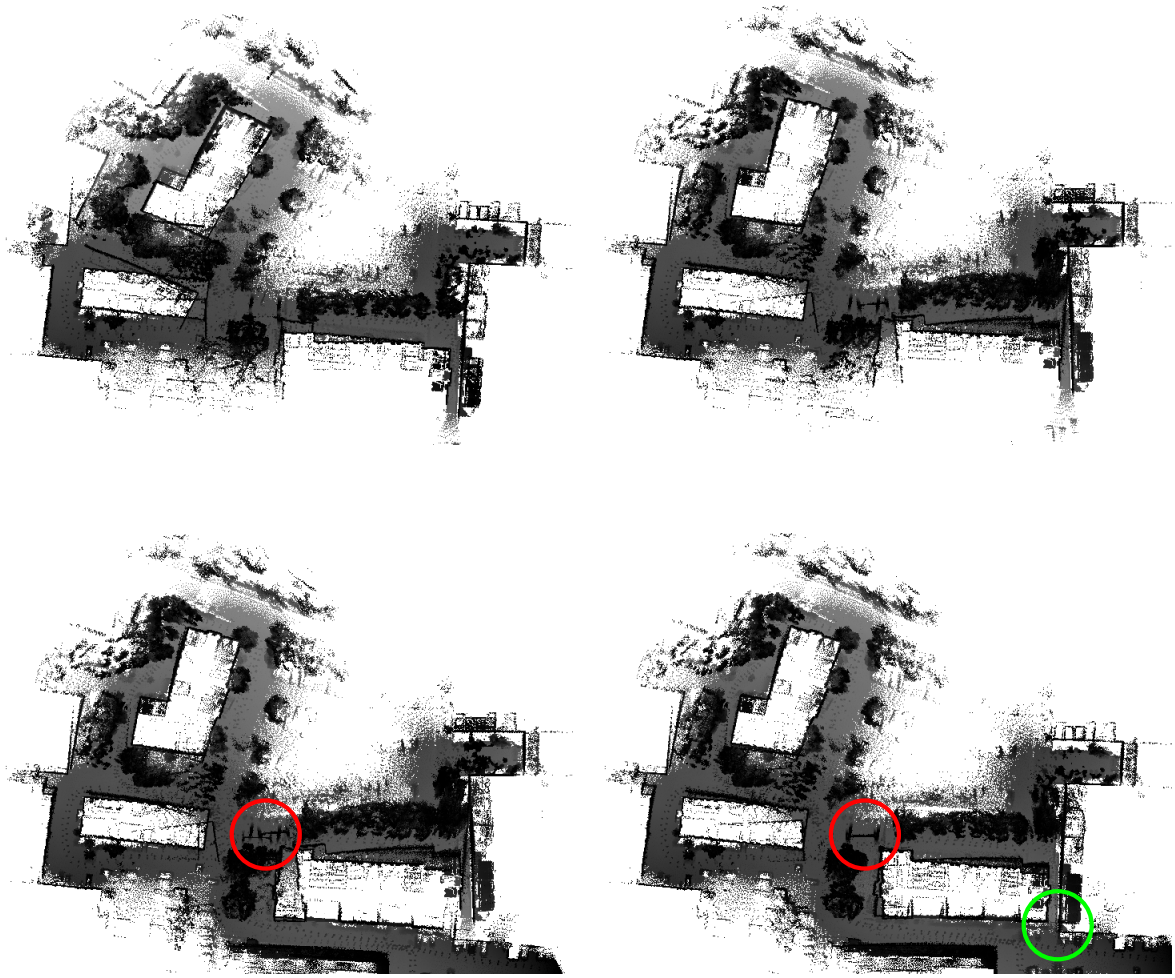


Abbildung 4.12: Ausschnitt aus dem Datensatz HANNOVER 1 links vor, rechts nach der der ELCH-Korrektur. Oben: mit *a* markierte Schleife aus Abbildung 4.13. Unten: Korrektur der in Abbildung 4.13 mit *b* markierten Schleife. Dabei wird die mit dem roten Kreis markierte Struktur (siehe Abbildung 1.1) richtig zusammengesetzt und die grün markierte Stelle passt so gut zusammen, dass ein rechter Winkel entsteht, obwohl an der Stelle keine Schleife geschlossen wird.

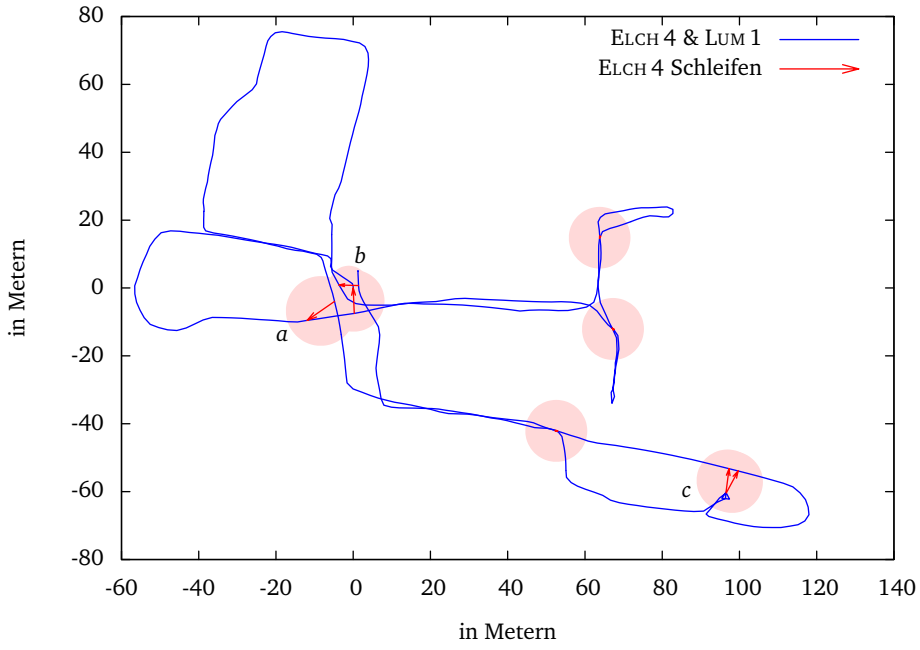


Abbildung 4.13: SLAM-Graph von ELCH 4 für den Datensatzes HANNOVER 1 bestehend aus Trajektorie (blau) und den gefundenen Schleifen (rote Pfeile, rosa hinterlegt), von denen man teilweise kaum etwas erkennen kann, da die gefahrenen Strecken aufeinander liegen. Markierung *a* und *b* zeigen die in Abbildung 4.12 zu sehenden Korrekturen. Bei Markierung *c* wird die selbe Schleife zweimal geschlossen (sie Abschnitt 4.2).

Dies scheiterte aber daran, dass die initialen Poseschätzungen zu schlecht sind. Deswegen habe ich die Software so erweitert, dass sie mit den durch ELCH und LUM ermittelten Posen weiterrechnen kann. Nun habe ich die Scans abwechselnd mit ELCH und gegen die Referenzdaten gematcht, bis alle Scans mit ICP gegen die Karte matchten. Zwischenzeitlich musste ich dabei sogar ein paar Scans von Hand verschieben, da ICP sie beim Matchen sehr weit von ihrer wahren Pose verschoben hatte. Mit der so gewonnenen Trajektorie konnte ich wie schon in Abbildung 4.6 den Fehler jedes Scans plotten (Abbildung 4.18).

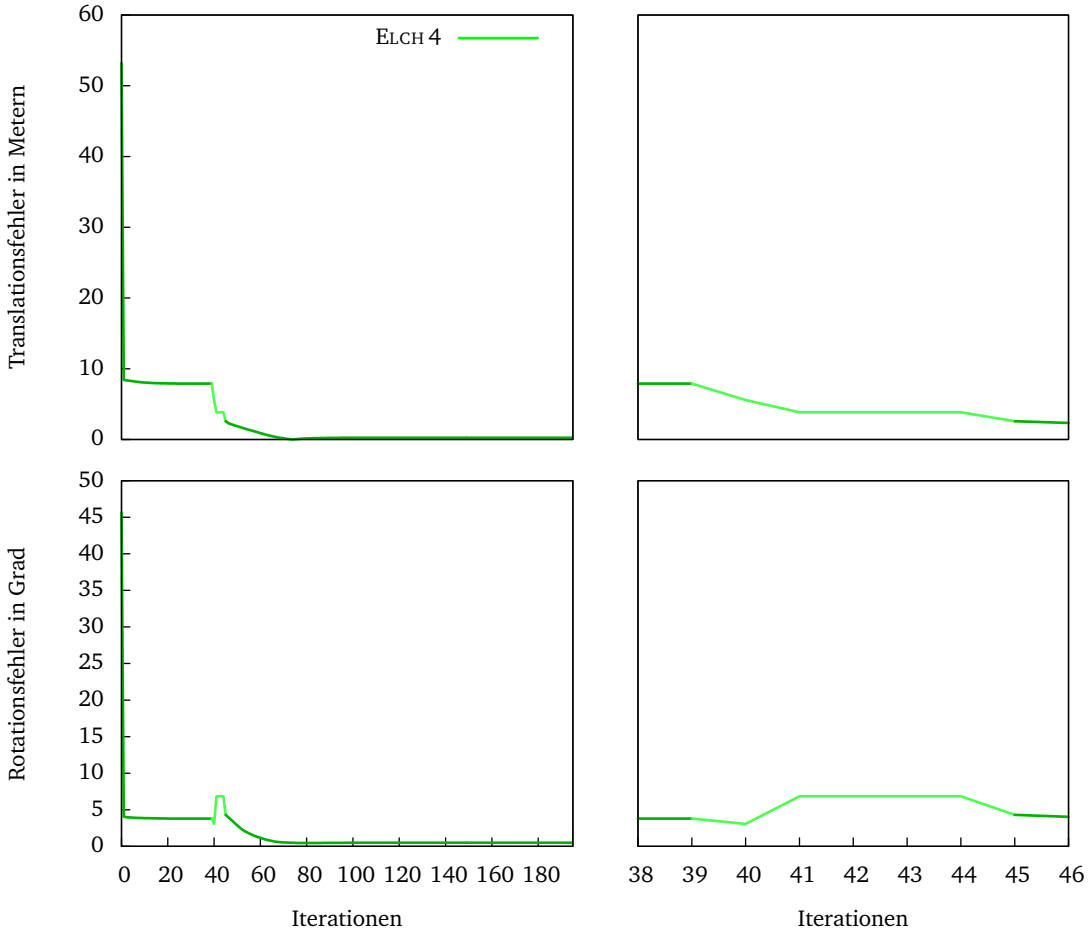


Abbildung 4.14: Konvergenzgraph des 200-ten Scans aus HANNOVER 1 (in Abbildung 4.11 bei Markierung *A* auf dem Weg von *F* nach *B*). Wie in Abbildung 4.12 zu sehen wird, der Rotationsfehler bei der zweiten ELCH-Korrektur zunächst schlechter, was aber bei einem weiteren Durchlauf wieder korrigiert wird.

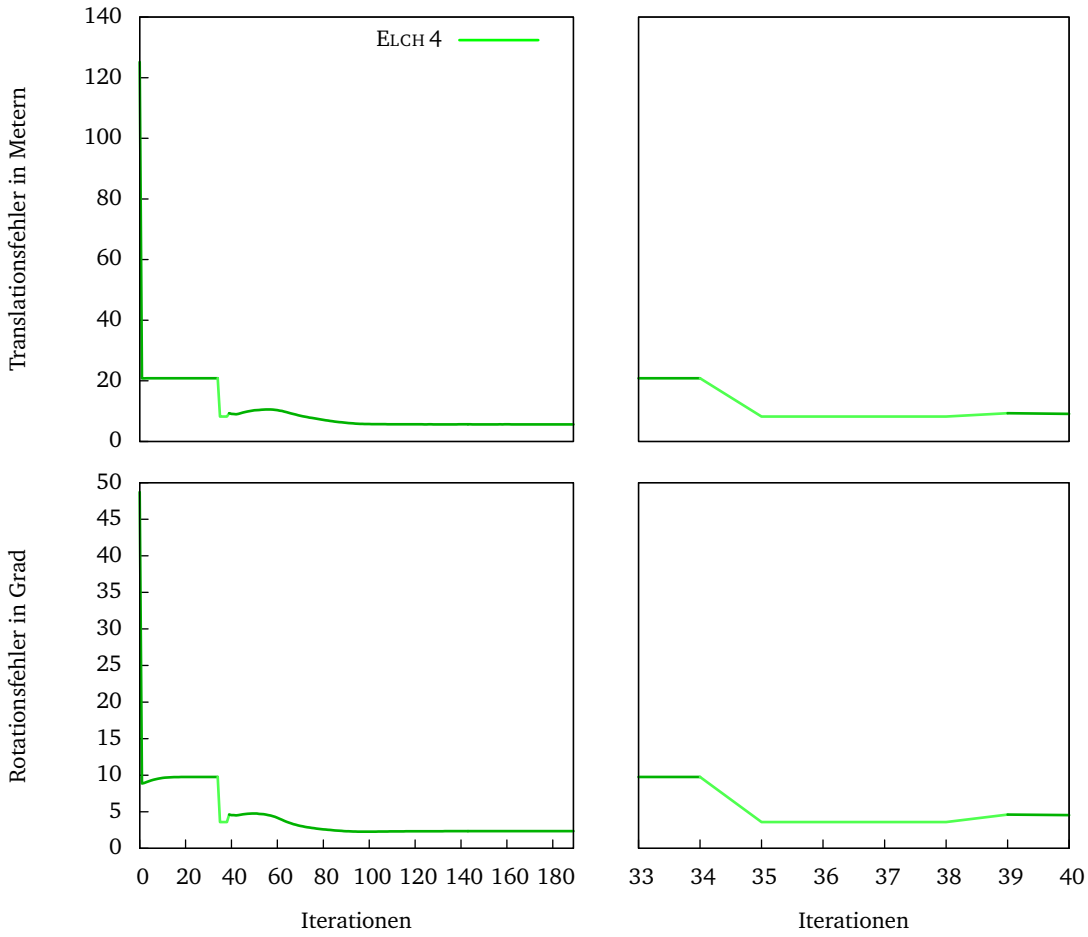


Abbildung 4.15: Konvergenz des 245-ten Scans (gelb eingefärbter Scan bei *G* in Abbildung 4.11).

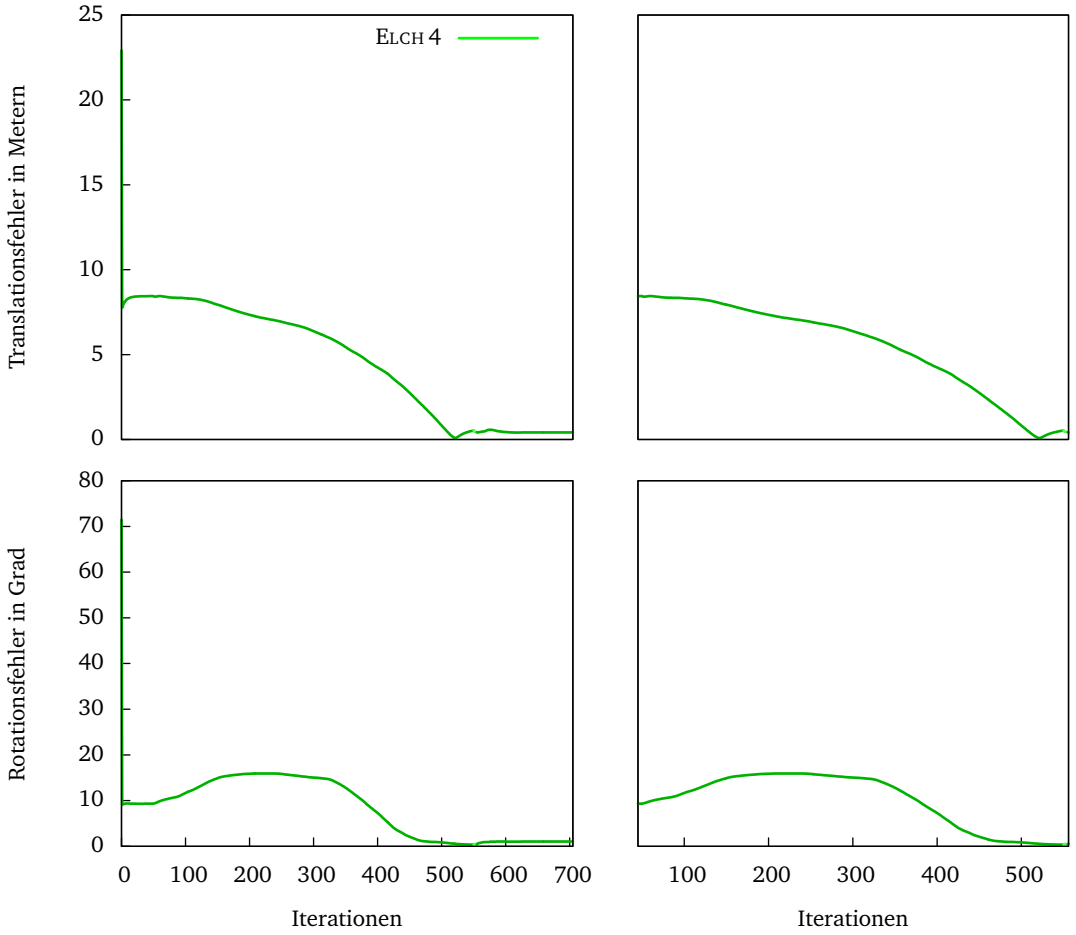


Abbildung 4.16: Der 308-te Scan (in Abbildung 4.11 bei *A* auf dem Weg von *B* nach *H*). Ab Iteration 45 wird per ICP der Korrekturvektor für ELCH berechnet.

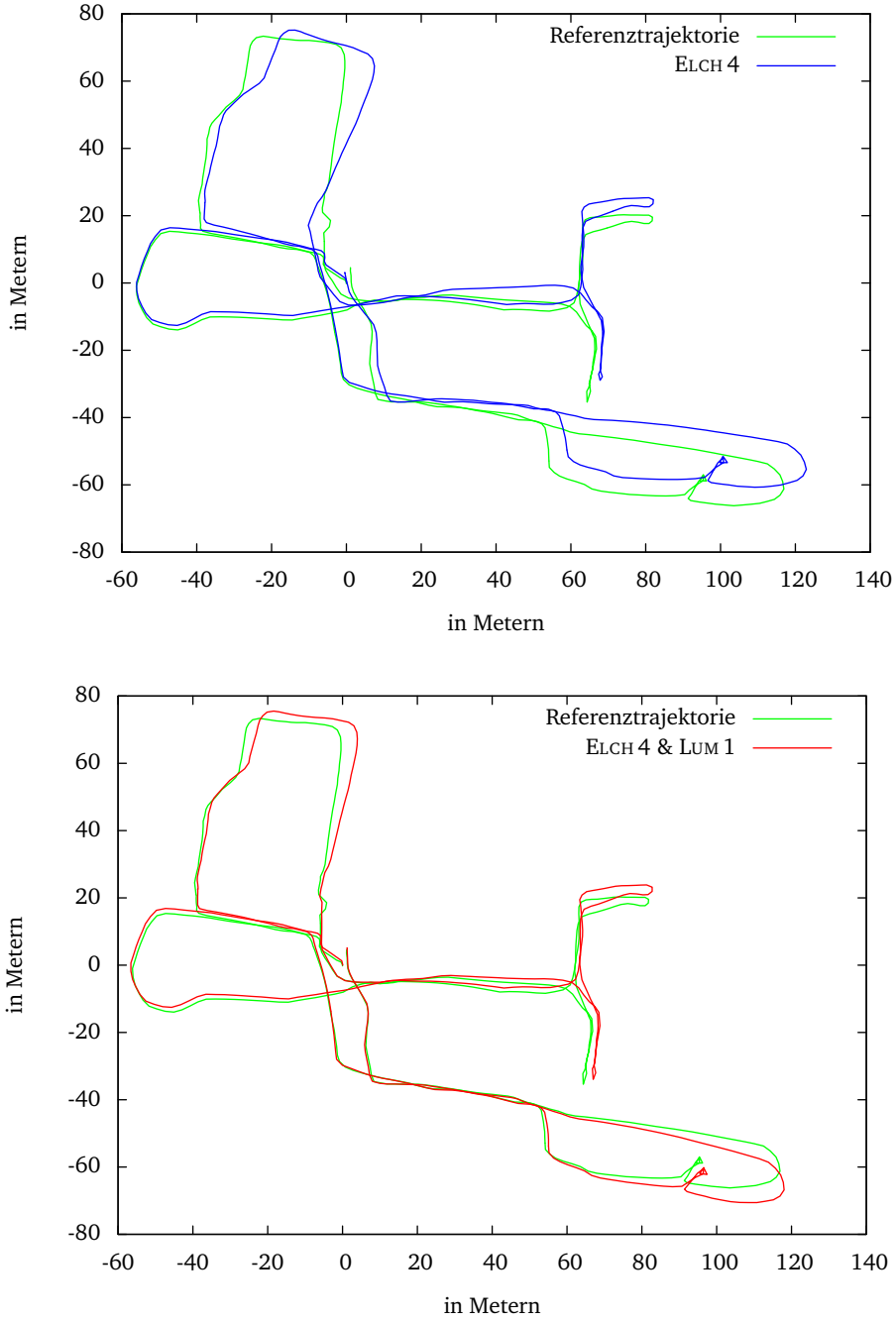


Abbildung 4.17: Die Trajektorie von HANNOVER 1 mit ELCH 4 (oben, blau) bzw. ELCH 4 und LUM 1 (unten, rot) verglichen mit der Referenztrajektorie (grün). Man kann erkennen, dass LUM die Schleife unten rechts im Vergleich zur Schleife darüber sehr viel mehr verschiebt und damit den in Abbildung 4.12 im grünen Kreis markierten rechten Winkel wieder kaputt macht.

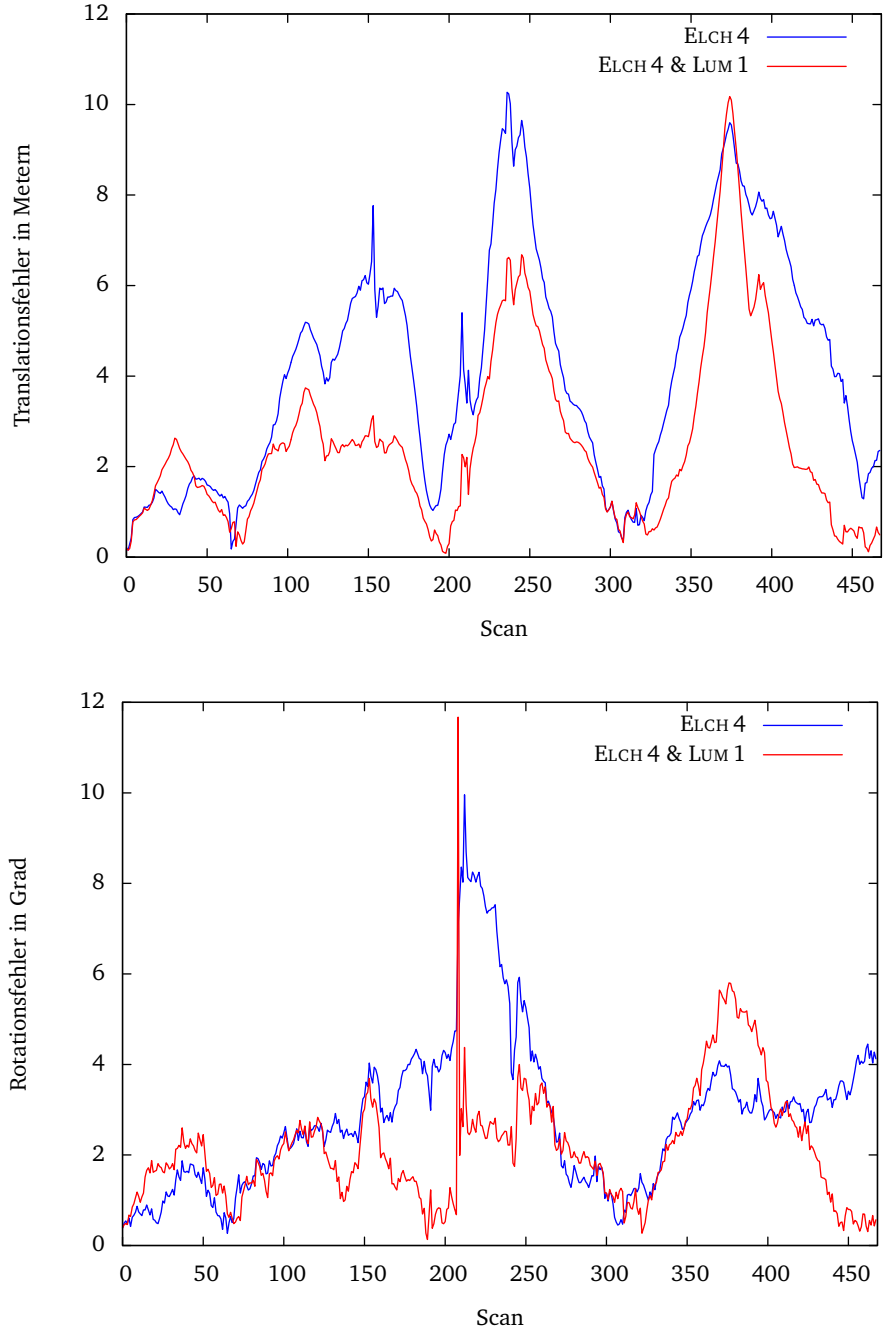


Abbildung 4.18: Translations- und Rotationsfehler von HANNOVER 1. Der Hauptfehler liegt wie bei HANNOVER 2 in der Höhenkoordinate (vergleiche Abbildung 4.17 und 4.6). Über alle Scans ergeben sich folgende Mittelwerte und Standardabweichungen: Translation: ELCH 4: (4,13 ± 2,57) m, ELCH 4 & LUM 1: (2,50 ± 1,97) m; Rotation: ELCH 4: (2,88 ± 1,72) °, ELCH 4 & LUM 1: (2,10 ± 1,30) °.

4.3 Der kombinierte Datensatz

Um die Idee zweier unabhängiger Roboterfahrten in der selben Umgebung aus Abschnitt 3.5 zu testen, habe ich beide bisher genannten Datensätze hintereinander gehängt (Abbildung 4.19). Die Software verarbeitet dabei erst den großen Datensatz und schließt alle Schleifen. Wenn sie damit fertig ist, wird die Pose auf die ungefähre Anfangspose des zweiten Datensatzes gesetzt und **keine** Kante im SLAM-Graph hinzugefügt. Der Anfang des neuen Datensatzes wird nun als Ende einer Schleife erkannt und gegen den alten Datensatz gematcht, wie man in Abbildung 4.20 erkennen kann. Mit dem gefundenen Fehler wird der neue Datensatz gegen den alten verschoben und eine Kante im SLAM-Graphen hinzugefügt. Beide Datensätze sind nun zu einem großen Datensatz verwoben und alle weiteren Schleifen können ganz normal geschlossen werden. Dabei kann es sowohl passieren, dass eine Schleife zwischen beiden Datensätzen entsteht, wie in Abbildung 4.21, also auch dass die Schleife im zweiten Datensatz beginnt und endet. Dies hat zwar zur Folge, dass beide Trajektorien zunächst nicht übereinander liegen, was aber von LUM korrigiert wird (siehe Abbildung 4.22). Durch eine geschicktere Wahl der Parameter oder bessere Schleifenerkennung sollte es möglich sein, auch diesen Fehler mit ELCH zu korrigieren. Durch die schlechte Schleifenerkennung war ich außerdem gezwungen, die Start- und Endwerte für zwei Schleifen von Hand zu setzen, damit sie zueinander passen. Die entsprechenden Werte habe ich dabei aus einem Einzellauf entnommen.

Um das Ergebnis bewerten zu können, habe ich wieder den Translations- und Rotationsfehler jedes einzelnen Scans ausgerechnet (Abbildung 4.23). Wenn man diese mit den einzelnen Experimenten

Abbildung 4.19: Die Karte der zusammengesetzten Datensätze HANNOVER 1 und HANNOVER 2. Für die Abfolge auf der Trajektorie vergleiche Abbildung 4.2 und 4.11.

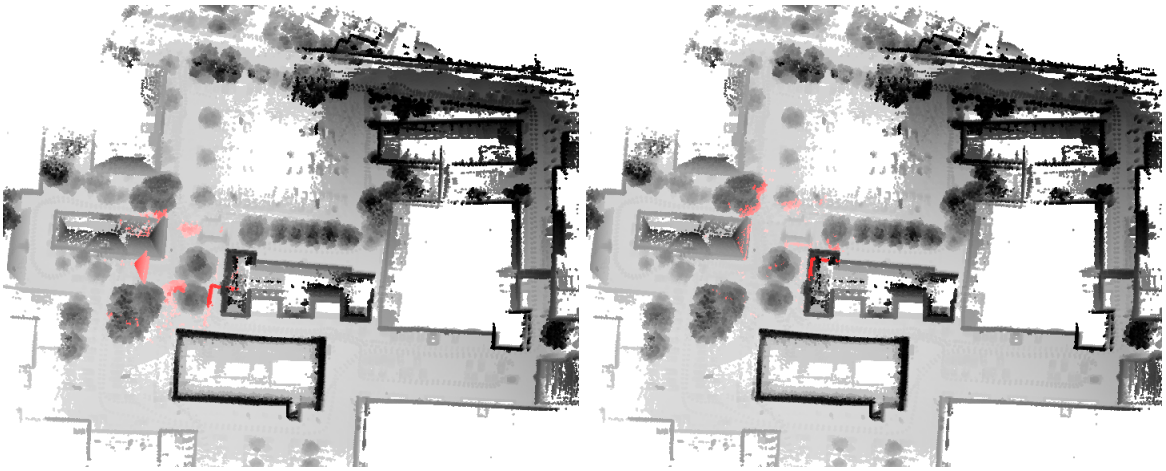


Abbildung 4.20: Registrierung des Anfangs von HANNOVER 1 und HANNOVER 2. Links: Initiale Poseschätzung (rot). Rechts: gematchte Pose.

Abbildung 4.21: ELCH-Korrektur der Häuser im roten Rechteck. Dabei wird auch der Baum im grünen Kreis aufeinander gezogen.

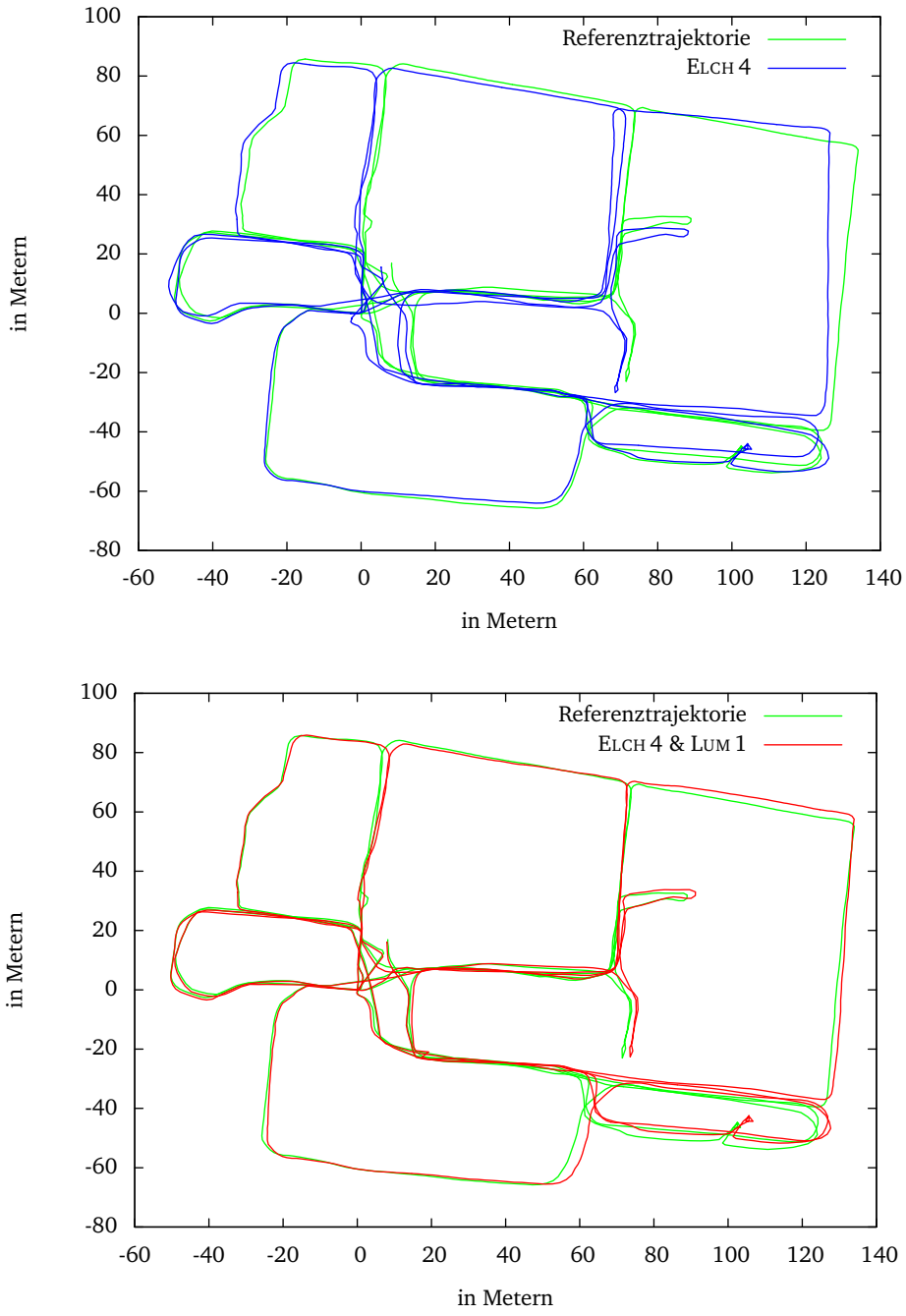


Abbildung 4.22: Trajektorien des zusammengesetzten Datensatzes nach ELCH bzw. LUM.

vergleicht (Abbildung 4.6 und 4.18), so kann man erkennen, dass die Maximalfehler kleiner werden. Dies ist auch an den Mittelwerten und Standardabweichungen zu erkennen, welche unter den jeweiligen Abbildungen stehen.

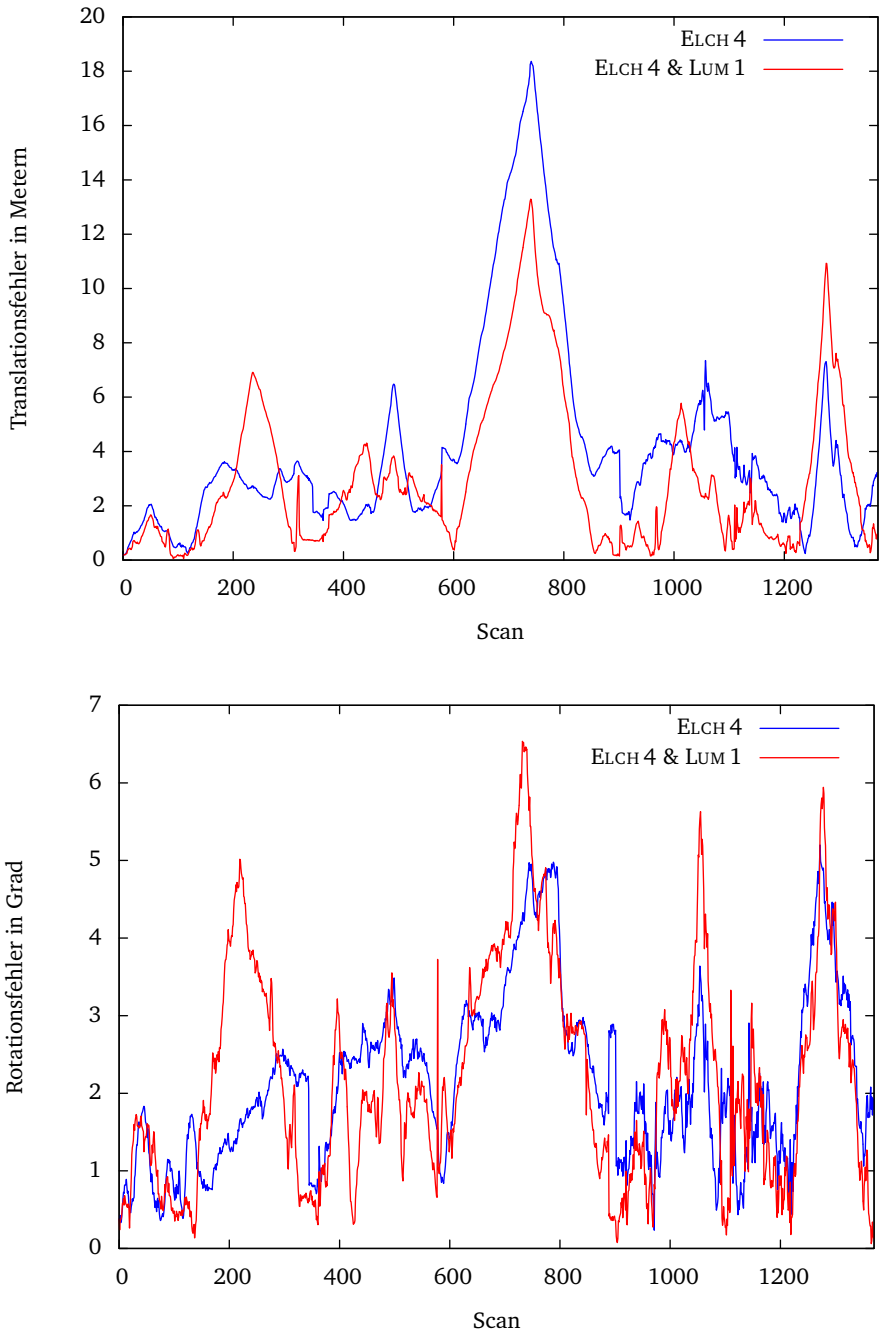


Abbildung 4.23: Translations- und Rotationsfehler des zusammengesetzten Datensatzes. Über alle Scans ergeben sich folgende Mittelwerte und Standardabweichungen: Translation: ELCH 4: (4,19 ± 3,7) m, ELCH 4 & LUM 1: (3,01 ± 2,82) m; Rotation: ELCH 4: (2,16 ± 1,08) °, ELCH 4 & LUM 1: (2,21 ± 1,4) °.

5 Zusammenfassung und Ausblick

Die in meiner Diplomarbeit entwickelte ELCH-Strategie ermöglicht deutlich bessere Ergebnisse bei einem Bruchteil der Laufzeit der bisherigen Algorithmen in der SLAM-Software. Dazu verbinde ich die lokalen Korrekturen von ICP mit einer globalen Fehlerverteilung, wobei ich nur die Fehler einiger spezieller Scans, nämlich die der Schleifenenden, betrachte. Diese lassen sich einfach feststellen und approximieren den Gesamtfehler, der auf einer Schleife gemacht wurde. Der LOA-Algorithmus schätzt danach über die Kovarianzen Gewichte für die komplette Robotertrajektorie ab, womit der gefundene Fehler verteilt wird.

Trotz der genannten Vorteile gibt es aber auch einige Möglichkeiten, den SLAM-Prozess noch schneller und besser zu machen. Ein Hauptproblem ist sicher die sehr naive Schleifenerkennung. Mit Ansätzen, die Merkmale in Laserscans berechnen und diese wiedererkennen oder gleich matchen, würde man eine deutlich robustere Schleifenerkennung bekommen. So könnte man besser Anfang und Ende einer Schleife finden und auch Schleifen mit einem sehr großen Fehler erkennen, wo Anfang und Ende sehr weit auseinander liegen. Auch die Berechnung der Varianzen ließe sich noch verbessern und beschleunigen, indem nicht die komplette Kovarianzmatrix errechnet und die Berechnung parallelisiert wird. Insgesamt kann man die ELCH-Strategie als ein Framework ansehen, bei dem man ohne viel Aufwand einzelne Teile durch bessere ersetzen, oder alternative Implementierungen testen kann, wie ich in Abschnitt 3.2 gezeigt habe.

Das in der Einleitung angesprochene Ziel der Kartenerstellung in Echtzeit auf dem Roboter ist durch ELCH zwar noch nicht erreicht, aber in greifbare Nähre gerückt. Die Zeit, um alle Scans zu matchen, lag bei allen meinen Experimenten deutlich unter der Zeit, die der Roboter zum Aufnehmen gebraucht hat. Trotzdem würden die erstellten SLAM-Graphen irgendwann so groß, dass sie nicht mehr in Echtzeit korrigiert werden könnten (siehe Abbildung 3.7 Formel IX). Hier böte sich zunächst an, statt Dijkstra eine anderen Kürzeste-Wege-Algorithmus zu nutzen. Um die Strategie wirklich echtzeitfähig zu machen, ist aber auch ein divide & conquer Ansatz vorstellbar, bei dem nur bestimmte Teile des Graphen korrigiert werden und für die anderen Knoten der Fehler nur approximiert wird.

Dies sind allerdings alles Dinge, die zukünftig sicher Probleme bereiten werden, momentan aber nicht dringend sind. Die nächsten logischen Schritte wären z. B. ein Vergleich mit den in Abschnitt 2.6 vorgestellten Ansätzen von Olson und Grisetti, was allerdings auf Grund des unterschiedlichen SLAM-Graphen nicht trivial ist. Eine andere Möglichkeit wäre die Erweiterung des Experiments mit mehreren Robotern aus Abschnitt 4.3. So könnten man z. B. abwechselnd von jedem Datensatz einen Scan zur Karte hinzufügen und so ein paralleles Fahren beider Roboter simulieren. Dies könnte man erweitern, indem man noch mehr Datensätze aufnimmt oder einen Datensatz teilt und diese alle gleichzeitig registriert.

Eine Frage, die ich in meiner Arbeit bewusst ausgeklammert habe, ist, wie weit sich ELCH der optimalen Lösung nähert. Dazu könnte man ihn z. B. mit dem Kalman-Filter vergleichen, der unter bestimmten Bedingungen optimal ist (Thrun et al. [31]).

6 Danksagung

Ich danke Herrn Professor Hertzberg, als Leiter der Arbeitsgruppe, und Andreas Nüchter und Kai Lingemann für viele interessante Diskussionen und Ideen für meinen Algorithmus, die Vorarbeiten in der SLAM-Software und Kommentare und Anregungen für diese Arbeit. Weiter danke ich Dorit Borrmann und Jan Elseberg, die den LUM-Algorithmus in der SLAM-Software implementiert haben und mir auch immer bei Fragen geholfen haben. Danken möchte ich auch Christof Söger, der sich immer Zeit für mich genommen hat, wenn ich an einer Stelle nicht weiter wusste, und meinen Freunden und Kommilitonen, die mir zugehört haben, wenn ich von meiner Arbeit erzählt habe. Ganz besonders möchte ich meinem Vater danken, mit dem ich über alles diskutieren konnte und der sehr viele Ideen und Anregungen beigetragen hat.

Literaturverzeichnis

[1] Arun, K. S. ; Huang, T. S. ; Blostein, S. D.: Least-Squares Fitting of Two 3-D Point Sets. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 9 (1987), Nr. 5, S. 698–700

[2] Bailey, T. ; Durrant-Whyte, H.: Simultaneous Localization and Mapping (Slam): Part II State of the Art. In: *IEEE Robotics & Automation Magazine* 13 (2006), Nr. 3, S. 108–117

[3] Besl, P. ; McKay, N.: A Method for Registration of 3–D Shapes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14 (1992), Nr. 2, S. 239–256

[4] Borrmann, D. ; Elseberg, J.: *Global konsistente 3D Kartierung am Beispiel des Botanischen Gartens in Osnabrück*. Osnabrück, Germany : Bachelor's thesis, 2006

[5] Borrmann, D. ; Elseberg, J. ; Lingemann, K. ; Nüchter, A. ; Hertzberg, J.: Globally Consistent 3D Mapping with Scan Matching. In: *Journal of Robotics and Autonomous Systems* 65 (2008), Nr. 2, S. 130–142

[6] Brucker, P. ; Knust, S.: *Complex Scheduling*. Springer, 2006

[7] Dijkstra, E. W.: A Note on Two Problems in Connexion with Graphs. In: *Numerische Mathematik* 1 (1959), Nr. 1, S. 269–271

[8] Durrant-Whyte, H. ; Bailey, T.: Simultaneous Localisation and Mapping (Slam): Part I The Essential Algorithms. In: *IEEE Robotics & Automation Magazine* 13 (2006), Nr. 2, S. 99–110

[9] Folkesson, J. ; Christensen, H.: Graphical Slam – A Self-Correcting Map. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004, S. 51–70

[10] Freese, U.: Efficient 6-Dof Slam with Treemap as a Generic Backend. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007, S. 4814–4819

[11] Grisetti, G. ; Grzonka, S. ; Stachniss, C. ; Pfaff, P. ; Burgard, W.: Efficient Estimation of Accurate Maximum Likelihood Maps in 3D. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, S. 3472–3478

[12] Gutmann, J.-S. ; Konolige, K.: Incremental Mapping of Large Cyclic Environments. In: *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 2000, S. 318–325

[13] Hertzberg, J. ; Lingemann, K. ; Nüchter, A.: *Mobile Roboter – Eine Einführung aus Sicht der Informatik*. Springer, 2009. – To appear

[14] Konolige, K.: Large-Scale Map-Making. In: *Proceedings of the National Conference on AI*, 2004, S. 457–463

[15] KUFFNER, J. J.: Effective Sampling and Distance Metrics for 3D Rigid Body Path Planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation* Bd. 4, 2004, S. 3993–3998

[16] LU, F. ; MILIOS, E.: Globally Consistent Range Scan Alignment for Environment Mapping. In: *Autonomous Robots* 4 (1997), Nr. 4, S. 333–349

[17] NEWMAN, P. ; COLE, D. ; HO, K.: Outdoor SLAM using Visual Appearance and Laser Ranging. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, 2006, S. 1180–1187

[18] NEWMAN, P. ; HO, K.: SLAM-Loop Closing with Visually Salient Features. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005, S. 635–642

[19] NIETO, J. ; BAILEY, T. ; NEBOT, E.: Recursive scan-matching SLAM. In: *Robotics and Autonomous Systems* 55 (2007), Nr. 1, S. 39–49

[20] NÜCHTER, A.: *3D Scan Repository*. `http://kos.informatik.uni-osnabrueck.de/3Dscans/`, 2009

[21] NÜCHTER, A. ; LINGEMANN, K.: SLAM *Software*. `http://slam6d.sourceforge.net/`, 2009

[22] NÜCHTER, A. ; LINGEMANN, K. ; HERTZBERG, J. ; SURMANN, H.: 6D SLAM with Approximate Data Association. In: *Proceedings of the IEEE International Conference on Advanced Robotics*, 2005, S. 242–249

[23] NÜCHTER, A. ; LINGEMANN, K. ; HERTZBERG, J. ; SURMANN, H.: 6D SLAM – 3D Mapping Outdoor Environments. In: *Journal of Field Robotics, Special Issue on Quantitative Performance Evaluation of Robotic and Intelligent Systems* 24 (2007), Nr. 8/9, S. 699–722

[24] OLSON, E.: *Robust and Efficient Robotic Mapping*. Cambridge, MA, USA, Massachusetts Institute of Technology, Diss., 2008

[25] OLSON, E. ; LEONARD, J. ; TELLER, S.: Fast Iterative Alignment of Pose Graphs with Poor Initial Estimates. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, 2006, S. 2262–2269

[26] PAZ, L. M.: *Divide and Conquer:* EKF SLAM *in* $\mathcal{O}(n)$. Zaragoza, Spain, Universidad de Zaragoza, Diss., 2008

[27] SHOEMAKE, K.: Animating Rotation with Quaternion Curves. In: *ACM SIGGRAPH Computer Graphics* 19 (1985), Nr. 3, S. 245–254

[28] SIEK, J. ; LEE, L. ; LUMSDAINE, A.: *Boost Graph Library*. `http://www.boost.org/libs/graph/`, 2009

[29] SPRICKERHOF, J.: ELCH *Website*. `http://jochen.sprickerhof.de/software/elch/`, 2009

[30] SPRICKERHOF, J. ; NÜCHTER, A. ; LINGEMANN, K. ; HERTZBERG, J.: An Explicit Loop Closing Technique for 6D SLAM. In: *Proceedings of the European Conference on Mobile Robotics*, 2009. – To appear

[31] THRUN, S. ; BURGARD, W. ; FOX, D.: *Probabilistic Robotics*. MIT Press, 2005

[32] WULF, O. ; BRENNEKE, C. ; WAGNER, B.: Colored 2D Maps for Robot Navigation with 3D Sensor Data. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 3, 2004, S. 2991–2996

[33] WULF, O. ; NÜCHTER, A. ; HERTZBERG, J. ; WAGNER, B.: Benchmarking Urban Six-Degree-of-Freedom Simultaneous Localization and Mapping. In: *Journal of Field Robotics* 25 (2008), Nr. 3, S. 148–163

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Osnabrück, 22. August 2009