Diplomarbeit

Graph-basierte 3D-Kartierung von Innenräumen mit einem RGBD-Multikamera-System

Erstellt von Stephan Stroucken
Vorgelegt am 18. Februar 2013

Unter Anleitung von Dipl. Inf. Jörg Stückler
Erstgutachter: Prof. Dr. Sven Behnke
Zweitgutachter: Dr. Dirk Schulz

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik VI
Autonome Intelligente Systeme

Zusammenfassung

Visuelle SLAM-Verfahren müssen dem Umstand Rechnung tragen, daß die Schätzungen von Trajektorie und Karte schwieriger werden wenn sich wenig Textur oder räumliche Struktur im Sichtbereich befinden. Ebenso ist es wünschenswert, den Speicherverbrauch und die Rechenzeit während des Kartiervorganges gering zu halten um größere Gebiete in einem Durchgang erfassen zu können.
Diese Arbeit zeigt einen Weg auf, vier RGBD-Kameras, die an einem gemeinsamen Träger fixiert sind, gleichzeitig zur Kartierung von Innenräumen zu benutzen. Durch die Anordnung im ca. 90° Winkel zwischen je zwei benachbarten Kameras wird ein vergrößerter Sichtbereich erzielt. Dieser erhöht die Wahrscheinlichkeit deutlich, ausreichend Struktur und Textur auch dort zu erfassen, wo eine Kamera allein suboptimale Bildinformationen liefern würde. Dies führt zu einer Steigerung der Robustheit des Kartierungsverfahrens. Zusätzlich wird eine Methode des Graph-Prunings eingeführt, die auf Basis einer "Multi Resolution Surfel Map"-Datenstruktur geeignete Knoten im SLAM-Graphen findet und zusammenfasst.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Wenn man eine Person um eine Wegbeschreibung bittet, könnte die Reaktion möglicherweise folgende sein: Die Person stellt sich den Startpunkt des Weges vor. Sie erinnert sich daran, in welche Richtung man losgehen muss, um zum Ziel zu gelangen. Im Geiste schreitet sie dann Stück für Stück den Weg ab und erinnert sich an markante Objekte, sowie deren Position relativ zu anderen Objekten. Während sie dem Fragesteller diese Informationen mitteilt("...immer geradeaus, an dem Stoppschild gegenüber des Supermarktes biegen sie dann links ab...") rekonstruiert sie also ein Abbild, oder eine Karte der Umgebung. Im Falle der Wegbeschreibung kann dies noch eine zweidimensionale Karte sein, während eine Person bei Aussagen wie "Der Schlüssel befindet sich in dem blauen Schränkchen unter dem Tisch gleich links neben der Tür, in der obersten Schublade ganz hinten." sicher an ein dreidimensionales Umgebungsmodell denkt.
Menschen arbeiten mit solchen Umgebungsinformationen täglich, intuitiv und ganz selbstverständlich. In vielen Bereichen informationsverarbeitender Wissenschaften, insbesondere in der Robotik, ist man bemüht diese Informationen(Karten) nach Möglichkeit korrekt und mit beschränktem Zeitaufwand digital erzeugen und ablegen zu können. Dort kann man auf Basis solcher Karten z.B. Pfadplanung und Greifplanung durchführen.
Auch könnte man sich vielleicht Folgendes vorstellen:
Ein potentieller Möbelkäufer hat sich auf der Internetseite eines Möbelfabrikanten ein neues Sofa ausgesucht. Da er sich ausserdem noch nicht in Gänze vorstellen kann, ob das Sofa optisch gut zu seinen anderen Möbeln passt, und er sich das Vermessen von Hand sparen möchte, macht er Folgendes:
Da er einen geeigneten optischen Sensor besitzt und der Möberhersteller die passende Software für ihn bereitstellt, erstellt er damit eine 3D-Karte seines Wohnzimmers. Dann lädt er sich, das ebenfalls vom Möbelhersteller bereits angefertigte, 3D-Modell des gewünschten Sofas herunter und fügt es virtuell

in sein Zimmer ein. So kann er bereits vor dem Kauf einen Eindruck erhalten ob ihm die Kombination gefällt, und welchen Platz das neue Möbelstück vielleicht einmal einnehmen soll.
Die Wahrnehmung der Umgebung ist schon über verschiedene Sensorenarten erreicht worden. Ultraschall, Laserscanner, Time Of Flight Kameras, Stereokameras und RGB-D-Kameras sind nur einige davon. Letztgenannte sind seit der Markteinführung von Microsofts X-Box 360° und der dazu erhältlichen Kinect Kamera sehr populär(Verkaufszahlen $> 10^7$ bis zum 26.03.2011 [12]) und preislich auch für den Massenmarkt erschwinglich geworden(z.B. 99,95€ bei Conrad [31]). Sie liefern Bildpaare, bestehend aus je einem RGB-Bild und einem dazugehörigen Tiefenbild, wobei mittels aktiver Infrarotmessung jedem Pixel eine Entfernung zur Kamera zugewiesen wird. Die zurzeit höchstmöglichen Auflösungen dieser Kameras belaufen sich auf 1280 x 1024(RGB-Bild), bzw. 640 x 480(Tiefenbild). Das Limit für die Aufnahmefrequenz unter voller Auflösung ist derzeit bei 30Hz. Diese Arbeit befasst sich mit der Erstellung von 3D-Karten von Innenräumen mittels solcher Kameras. Der Fokus liegt dabei auf der Verwendung mehrerer Kameras gleichzeitig, sowie der Eingrenzung des dabei benötigten Hauptspeichers.

1.1 Motivation

Man stelle sich vor, man steht vor einer weißen Wand und blickt senkrecht darauf. Die im Sichtfeld befindliche Fläche ist komplett weiß. Nun findet eine Bewegung parallel zur Wand statt. Wenn die Wand breit genug ist, dann ist das, was während der kompletten Bewegung gesehen wurde, unverändert. Als Mensch weiß man zwar, dass man sich bewegt hat, aber allein an der visuellen Wahrnehmung ist das nicht festzumachen. Dasselbe Problem hat eine RGBD-Kamera in dieser Situation - es ist allein anhand der aufgenommenen Bilder zwischen Beginn und Ende der Bewegung nicht möglich, herauszufinden ob man sich bewegt hat oder nicht, weshalb solche Situationen zu den klassischen Fehlerquellen der visuellen Odometrie zählen. Logischerweise muss die Wand nicht weiß oder gar einfarbig sein. Alle Wände oder große Gegenstände mit wenig Struktur oder sich wiederholenden Strukturen wie z.B. Backsteinwände können Ursache dieses Problems sein.
Die simultane Verwendung mehrerer RGBD-Kameras kann helfen dieses Risiko zu vermindern. Wenn die Kameras in unterschiedliche Richtungen schauen, so ist die Wahrscheinlichkeit klein dass sich alle verwendeten Kameras gleichzeitig in einer solchen Situation befinden. So kann gegebenenfalls der Informationsverlust kompensiert werden.
Zur Repräsentation der Kartierung wird in dieser Arbeit ein topologischer

Graph, bestehend aus einer Knoten- und einer Kantenmenge, benutzt. Die Knoten des Graphen repräsentieren Raumpositionen mit sechs Freiheitsgraden, die von den Kameras bereits besucht worden sind. Kanten beinhalten die Unsicherheiten in der gegenseitigen Lage ihrer adjazenten Knoten. Die Knotenmenge eines solchen Graphen korreliert demnach positiv mit der Größe des kartierten Gebietes. Da zusätzlich zu jedem Knoten auch die dazugehörige Karteninformation abgespeichert wird, wächst der Speicherverbrauch während der Kartierung kontinuierlich an. Trotz immerfort steigender Speicher- und Leistungskapazitäten der verfügbaren Rechensysteme werden so auf jedem Rechensystem kritische Werte erreicht, wenn das kartierte Gebiet eine bestimmte Größe überschreitet. Desweiteren kann es bereits vor Annäherung an diese Grenzen wünschenswert sein, den Speicher- und Rechenbedarf zu reduzieren, falls die Energieversorgung begrenzt ist oder der Rechner zeitgleich noch andere Aufgaben ausführen soll. Daher zeigt diese Arbeit eine Methode des Graph-Pruning auf, mit welcher der Graph zur Laufzeit Knoten und Kanten verlieren kann und somit der verwendete Speicherbedarf reduziert werden kann.

Kapitel 2

Verwandte Arbeiten

Eine Kernfrage existierender Arbeiten zum Thema Simultaneous Localization and Mapping(SLAM) ist die nach der Art und dem Aufbau der Sensorik, verbunden mit den jeweiligen Vor- und Nachteilen. Ebenso widmet man sich häufig den Möglichkeiten um Speicherplatz und Rechenzeit zu sparen, der Detektion von Schleifenschlüssen(Loop Closings), der Erhöhung der Robustheit gegenüber Messfehlern und der Korrektur von Fehlern in der Karte.

2.1 Sensorik

2.1.1 Laser

3D-Laserscanner tasten ihre Umgebung hochfrequent ab und liefern meistens einzeln trennbare Laserscans in Form einer 3-D Punktwolke zurück.

Abbildung 2.1: Laserscanner

Die Aufnahmefrequenz, gepaart mit einem meist sehr großen Abtastwinkel haben zur Folge, dass zwei aufeinanderfolgende Laserscans in der Regel einen großen Überlappungsbereich haben, was deren Registrierung (Scanmatching) zueinander sehr begünstigt. Leider liefern Laserscanner in der Regel keine Farbinformation und aktuell steigt der Anschaffungspreis mit den Ansprüchen an die Genauigkeit stark an.
Hähnel, Burgard, Fox und Thrun verwendeten einen Laserscanner auf einem Roboter [15]. Sie haben einen Rao-Blackwell-Partikelfilter verwendet, um Laserscans in Schätzungen für Roboterposen umzuwandeln, geschlossene Schleifen zu erkennen und an Hand dessen die Trajektorie aus der Radodometrie entscheidend zu verbessern.
In [18], wurde ebenfalls ein Laser benutzt. Dort wurde ein Ansatz vorgestellt um Knoten zum Entfernen aus dem SLAM-Graphen auszuwählen, sowie die Information aus den entfernten Knoten auf bestehende Knoten zu verteilen.

2.1.2 Stereokameras

Eine Stereokamera besteht aus einem Kamerapaar mit stark überlappenden Sichtbereichen, die an einem gemeinsamen Träger angebracht sind.

Abbildung 2.2: Stereokamera

Eine gängige Gebrauchsweise ist, in einem Bildpaar Bereiche zu finden, die zum selben Objektpunkt gehören(z.B. über SIFT-, SURF- oder ORB-Features [21] [1] [29]). Ist der Abstand zwischen beiden Projektionszentren(Stereobasis), sowie die Parameter der inneren Orientierung bekannt, kann man über einen räumlichen Vorwärtsschnitt die Koordinaten korrekt identifizierter Objektpunkte berechnen.

In [17] wurde ein robustes Langzeitkartierungsverfahren für dynamische Umgebungen vorgestellt, bei dem die Experimente mit Stereokameras durchgeführt wurden.

2.1.3 Monokameras

Monokameras sind herkömmliche (RGB-)Kameras.

Abbildung 2.3: Monokamera

Ein Bild einer Monokamera ist eine zweidimensionale Projektion des dreidimensionalen Raumes, was den Informationsverlust über einen Freiheitsgrad zur Folge hat. Es können auch hier korrespondierende Punkte in zwei verschiedenen Bildern gefunden werden, aber durch den fehlenden Freiheitsgrad ist es nicht ohne weiteres möglich die dritte Dimension zu rekonstruieren. Dies soll durch folgendes Bild veranschaulicht werden:

Abbildung 2.4: Grosser Fuß oder Miniaturstonehenge? [5]

Ist der Fuß normal groß und befindet sich über dem Stonehenge? In diesem Fall kann es sich nur um ein Miniaturmodell desselben handeln. Man könnte auch schlussfolgern dass der Fuß, sowie auch das Stonehenge beide Originalgröße haben, aber der Fuß wesentlich näher an der Kamera dran ist. Theoretisch wäre allerdings beides möglich. Ein weiteres Beispiel:

Abbildung 2.5: Der Mond [36]

Kennt man den Erdtrabanten, sowie seine Größe und Entfernung zur Erde, so liegt es nahe, ihn in diesem Bild als die mittig platzierte Scheibe zu identifizieren. Eine Kamera verfügt nicht über derartige Kontextinformation. Für sie macht es keinen Unterschied, ob das abgebildete Objekt groß und weit weg(Mond), oder klein und nah(ein durch das Bild geworfener Ball) ist. Anders ausgedrückt: In einem 2D-Bild ist die Größe eines Objektes nicht von seiner Nähe zur Kamera zu unterscheiden.
Dennoch sind Monokameras unter bestimmten Bedingungen auch zur Kartierung geeignet, wie Sven Lange in seiner Diplomarbeit [7] gezeigt hat. Befindet sich die Kamera z.B. an einem Objekt(oder fährt an einer Schiene), das sich gleichförmig bewegt, so können aus der Aufnahmefrequenz und der Geschwindigkeit Annahmen über die Transformation zwischen den Kamerapositionen bei benachbarten Aufnahmen gemacht, und damit analog zum Stereokamerafall verfahren werden. Besser ist die hier verwendete Passpunktvariante: Über den Kartierungsbereich verstreut befinden sich sog. Passpunkte, also Punkte deren Positionen in einem gemeinsamen Koordinatensystem vorab bekannt sind. Befinden sich in einem überlappenden Bildpaar solche Passpunkte, so kann man mithilfe derer das 3D-Modell rekonstruieren.

2.1.4 RGBD-Kameras

RGBD-Kameras sind doppelsensorische Kameras, die auf herkömmliche Art RGB-Bilder aufnehmen, und "zeitgleich"(= mit sehr geringem zeitlichem Versatz) zu jedem RGB-Bild ein zugehöriges Tiefenbild erzeugen.

Abbildung 2.6: RGBD-Kamera ASUS X-tion Pro Live

Ein Tiefenbild ist ein graustufiges Bild, bei dem die Helligkeitsabstufungen die Entfernungen zur Kamera codieren. Genaueres zur Funktionsweise findet sich im Kapitel "Sensorik ".
In [13] wurde ein Kartierungsverfahren vorgestellt das eine RGBD-Kamera verwendete. Als Besonderheit um RGBD-Bilder aufeinander zu registrieren stellen sie eine Abwandlung des ICP(Iterative Closest Point)-Algorithmus vor, den RGBD-ICP. Dieser berechnet zwei Transformationen, eine über den Original ICP-Algorithmus und eine durch das Extrahieren von SIFT-Merkmalen mit anschliessender Transformationsbestimmung mittels RANSAC (Random Sample Consensus). Die Ausgabe ist dann eine gewichtete Mittelung beider Transformationen. Diese Arbeit sucht ferner nach Schleifenschlüssen in einer bestimmtem Umgebung um die aktuelle Posenschätzung.
Michael Paton verfasste seine Masterarbeit [27] zum Thema RGBD-Kartierung. Er hat dort den GICP(General ICP) Algorithmus [32] implementiert, der ebenfalls eine Abwandlung des Original ICP-Algorithmus ist, und in der Fehlerfunktion nicht nur Punkt zu Punkt-Abstände, sondern auch Punkt-Ebene- und Ebene-Ebene-Abstände einbezieht. Diesen benutzt er in einer Abwandlung des RGBD-ICP-Verfahrens von [13], wobei er das Ergebnis aus Merkmalsextraktion und RANSAC als Initialschätzung für den GICP-Algorithmus benutzt. Eine Besonderheit ist hier, dass im Falle einer zu geringen Anzahl korrespondierender Bildmerkmale auf die Initialisierung mit RANSAC verzichtet wird.

2.1.5 Hybridansätze

Die Besonderheit einer Hybridlösung ist, dass mindestens zwei verschiedene Sensortypen mit zur Kartierung beitragen. Dies bietet die Möglichkeit, an Stellen, wo ein bestimmter Sensor Schwächen hat, ihn mit einem anderen zu ergänzen bzw. durch ihn zu ersetzen. Auch können verschiedene Sensoren sich um verschiedene Aufgaben während der Kartierung kümmern.
Zum Beispiel haben Newman, Cole und Ho in [24] einen mobilen Roboter mit einem Laserscanner und einer Kamera ausgestattet um ihn zur Kartierung zu verwenden. Die Posenschätzung läuft immer auf Basis der Lasermessung ab. Zusätzlich nimmt die Kamera alle paar Meter nach links und rechts jeweils ein Bild auf. Aus diesen Bildern werden die SIFT-Deskriptoren extrahiert und mit der dort geschätzten Roboterpose verknüpft. Auf Basis dieser SIFT-Merkmale wird dann nach Schleifenschlüssen gesucht.
Einen aussergewöhnlichen Weg sind Gallegos, Meilland, Rives und Comport in [10] gegangen. Sie haben einen mobilen Roboter mit einer omnidirektionalen 360° Kamera und einem Laserbereichscanner ausgestattet und beide zueinander kalibriert. Sie nutzen aus, dass senkrechte Kanten(z.B. an Fenstern, Türen, Wänden) in einem 360° Bild nahezu radial verlaufen. Also wird aus dem omnidirektionalen Bild ein Kantenbild erzeugt und mittels Hough-Transformation nach solchen Kanten gesucht. Da die Kalibrierung zwischen Laser und Kamera bekannt ist, kann man die Laserstrahlen in das Bild projizieren und an Schnittstellen zwischen Strahlenprojektion und radialen Kanten die Tiefe abgreifen, um die Bildinformation dort ins Dreidimensionale zu übertragen.

2.2 Graph-Pruning

Das Pruning des SLAM-Graphen, also das Entfernen von Knoten und Kanten zur Laufzeit, ist interessant und in vielen Fällen sogar notwendig, dank der damit verbundenen Einsparungen bezüglich Rechenzeit und Speicherplatz des verarbeitenden Rechners.

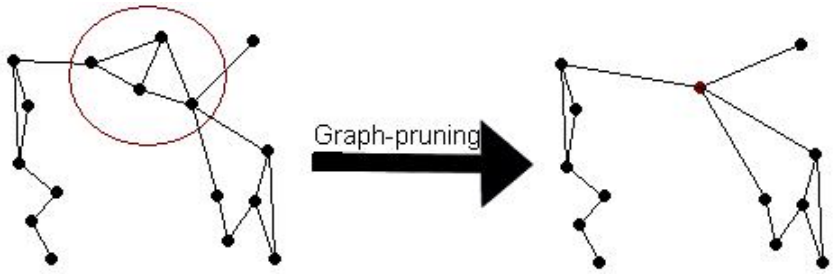


Abbildung 2.7: Beispiel für Graph-Pruning

Einige Sensoren liefern eine hohe Dichte an Karteninformationen, was in einem höheren Speicherbedarf pro Knoten resultieren kann. Berechnungen, wie z.B. das Loop Closing, haben häufig Laufzeiten, die in Relation zur Knotenmenge stehen. Möglich ist auch, dass die Energievorräte begrenzt sind(z.B. auf einem Marsroboter) und man deshalb Berechnungen, und damit Energie sparen möchte. Umstände wie diese haben dazu geführt, dass das Graph-Pruning(sowie Verfahren mit ähnlichen Effekten) zu einem der Kernthemen im Bereich des autonomen Kartenbaus geworden ist.
In [9] wurde ein hierarchischer Ansatz erarbeitet. Dieser sieht vor, dass die gesamte Karte in stochastisch unabhängige Teilkarten unterteilt ist. Ein neuer Kartenteil wird eröffnet, sobald der Informationsgehalt im aktuellen Kartenteil eine festgelegte Obergrenze erreicht, die Unsicherheit der aktuellen Posenschätzung relativ zum Ursprung der Teilkarte zu groß wird oder eine Sensormessung nicht mehr in das Kartenstück eingeordnet werden kann. Dies führt zu einem Zweischichtensystem in der Karte. In der ersten Schicht steht jeder Knoten für den Ursprung einer lokalen Teilkarte. Diese werden aber als Ganzes betrachtet und nicht weiter aufgelöst. In der zweiten Schicht repräsentiert die Knotenmenge, wie in den meisten anderen Verfahren üblich, eine Teilmenge aller Sensormessungen. Es fällt hier also die Gruppierung weg.

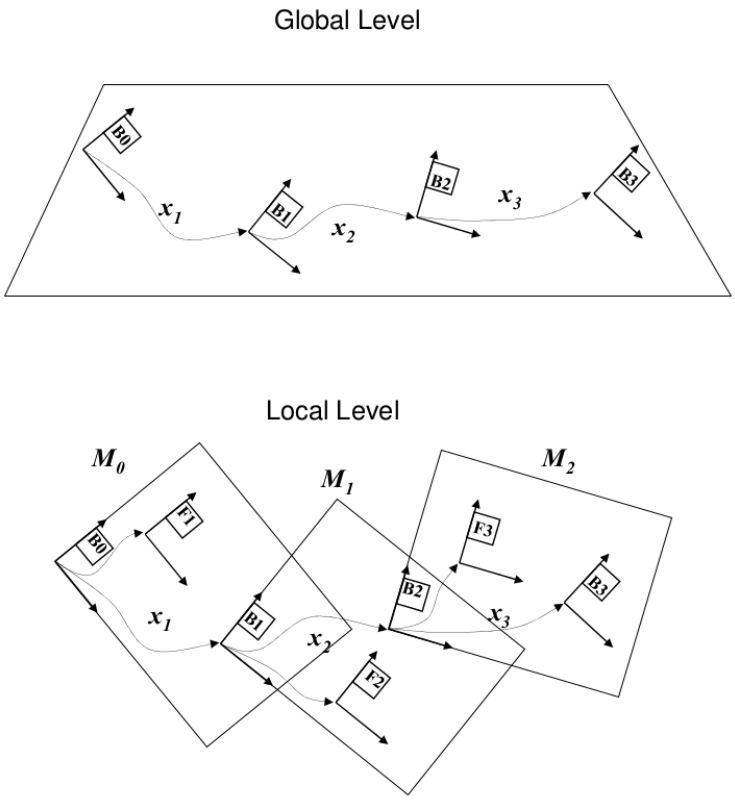


Abbildung 2.8: Zwei-Level-Repräsentation der Karte

Der Hauptbeitrag dieser Arbeit bezieht sich darauf, das Loop Closing nur in der ersten Schicht auszuführen, was durch die stark verringerte Graphdichte zu verkürzter Laufzeit führt. Eine Einsparung von Speicherplatz hierdurch wird in dieser Arbeit nicht erzielt.
Der Ansatz, die Karte in Teilkarten zu unterteilen, wurde auch in [26] benutzt. Hierbei wurde kein SLAM-, sondern ein SAM-System(Simultaneous Smoothing and Mapping) [8] implementiert. Der Unterschied besteht im Generellen darin, Posen von neuen Knoten im Graph nicht allein aus der Information aus Referenzknoten und aktueller Messung zu schätzen, sondern eine Glättung über die gesamte Trajektorie zu berechnen, so dass der neue Knoten bestmöglich eingepflegt wird. Hier wird SAM während der Erzeugung einzelner Kartenstücke eingesetzt, wobei nur über die lokale Trajektorie geglättet wird. Nach dem Abschluss des Kartenteiles wird die Trajektorie dann fixiert, unter der Annahme, dass zukünftige Änderungen an dieser ohne Fixierung nur minimal sein würden. Optimiert werden also nur Knoten im aktuell offenen Kartenstück, sowie alle geschlossenen Teilkarten untereinander.

Damit bringt diese Arbeit einen Beitrag, bezogen auf die Laufzeit des SAM-Verfahrens unter Beibehaltung der ursprünglichen Genauigkeitsansprüche.
Eine Arbeit, die den Graph tatsächlich verkleinert und damit auch Speicherplatz spart, ist [18], bzw. [19]. Sie besteht aus zwei Teilen. Der erste Teil beschäftigt sich mit einer Strategie, die Knoten zum Entfernen auswählt, deren relative Karteninformation(konkret: Der Betrag der Differenz aus Informationsgehalt der Gesamtkarte vor- und nach dem Entfernen des Knotens) möglichst klein ist. Der zweite Teil beschreibt, wie die Information aus allen, zu dem gelöschten Knoten inzidenten, Kanten auf verbleibende Kanten verteilt werden kann. Der naive Ansatz würde eine Clique erstellen, also zwischen allen Paaren von Knoten, die zu dem gelöschten Knoten adjazent sind, je eine Kante einziehen. Stattdessen beschreibt dieser Ansatz die Verwendung eines Chow-Liu-Baumes [4], der in diesem Fall dasselbe ist wie ein maximal gewichteter Spannbaum aus o.g. Clique. Die Gewichte der Kanten sind proportional zu dem Informationsgehalt, der in ihnen steckt. Experimentell getestet wurden zwei Strategien. Eine limitiert die Knotenzahl, d.h. der Prozeß setzt immer dann ein, wenn diese eine bestimmte Obergrenze erreicht. Die andere Strategie löscht grundsätzlich alle Knoten, deren relative Karteninformation(s.o.) unter einem Schwellwert liegen. Konkrete Angaben zur allgemeinen Ersparnis bezüglich Laufzeit- und Speicherkomplexität werden nicht gemacht. Angegeben sind allerdings Ergebnisse von Experimenten, z.B. 2597/15695(Knoten/Kanten ohne Pruning) ; 200/315(200 als Obergrenze der Knotenzahl) ; 148/250(setzen einer Untergrenze für den relativen Informationsgehalt). Im letzten Fall ergab sich als Folge des Löschprozesses, dass sich 1,6 Prozent aller Zellen in einer zugrundeliegenden Belegtheitsgitterkarte veränderten.

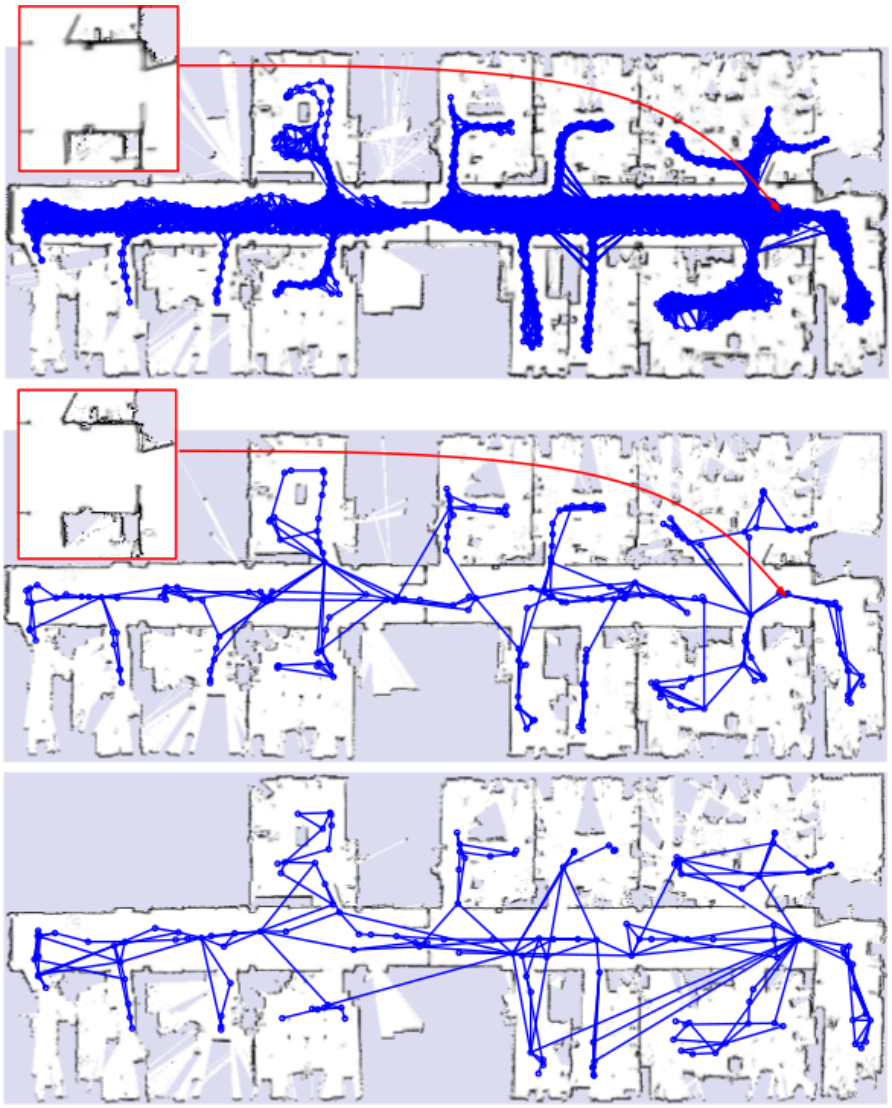


Abbildung 2.9: Oben: ohne Pruning, 2597 Knoten, 15695 Kanten; Mitte: Knotenzahl auf 200 begrenzt, 315 Kanten; Unten: Schwellwert für den relativen Informationsgehalt, 148 Knoten, 250 Kanten

2.3 Loop Closing

Wenn der, während eines Kartierungsvorganges zurückgelegte, Weg sich selber kreuzt, so wird dies auch Schleife oder Loop genannt. Die erfolgreiche Detektion solcher Schleifen wird analog als Schleifenschluss oder Loop Closure bezeichnet. Wo, und ob dies überhaupt passiert lässt sich ohne Weiteres nicht aus einer geschätzten Trajektorie ablesen, da Fehler in den Schätzungen

sich zu einem Drift aufakkumulieren. Das kann, insbesondere bei Fehlern bezüglich der geschätzten Winkel, dazu führen, dass die geschätzte Pose stark von der realen Pose abweicht. Könnte man zuverlässig Knotenpaare im SLAM-Graphen finden, die Orte repräsentieren, die in der realen Welt nah beieinander sind, so ist dies in aller Regel für die Genauigkeit der geschätzten Karte von großer Bedeutung. Aufgrund der Kenntnis, dass beide Knoten dicht zusammen sind, kann man ihre Positionen im Graph entsprechend anpassen. So entstehen stabilisierende Punkte im Graph, an denen hohe Posengenauigkeit herrscht und die als Folge den Posendrift umliegender Knoten bis hin zu größeren Entfernungen deutlich reduzieren können. So nützlich es sein kann diese Knotenpaare zuverlässig zu finden, so schwierig kann auch der Entwurf eines effizienten Algorithmus‘ sein, der dies leistet. Wenn ein Graph bereits ausgedehnt ist und viele Knoten enthält, ist es nicht mehr praktikabel, jeden neu eingefügten Knoten gegen alle anderen Knoten zu registrieren. Doch auch auf diesem Gebiet wurde bereits eine Menge Forschungsarbeit geleistet, von der ein Teil im Folgenden vorgestellt wird. Es sei an dieser Stelle jedoch erwähnt, dass die Erkennung von Loop Closures nicht Bestandteil dieser Arbeit ist und verwandte Arbeiten hierzu nur deshalb zitiert werden, weil es eine hohe Relevanz zum Thema Kartenbau besitzt.

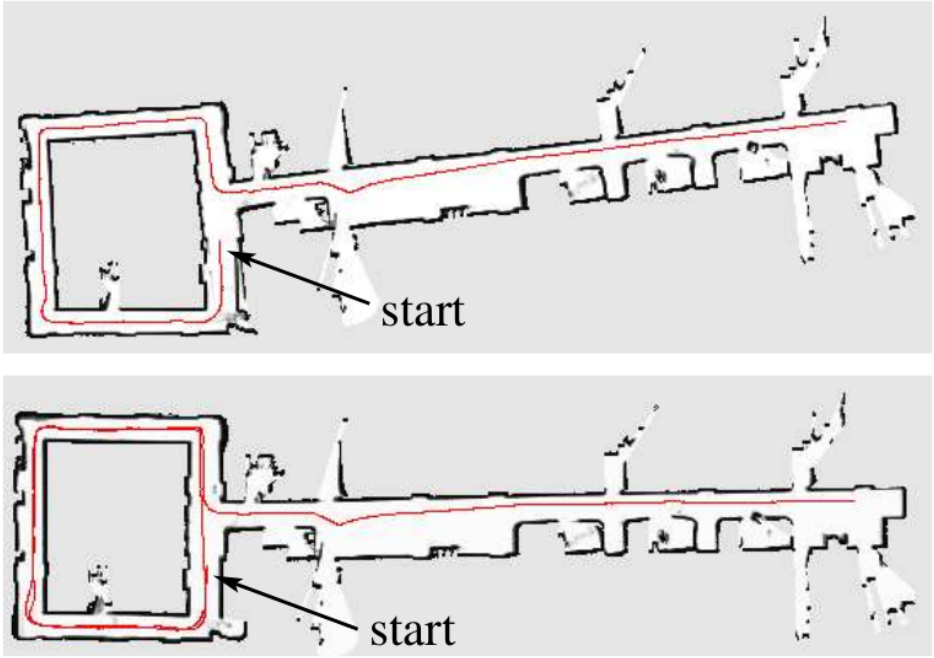


Abbildung 2.10: Loop Closing Beispielbild: Ein Rundweg mit angrenzendem Korridor in der Sieg Hall(University of Washington). Oben: Ohne Loop Closing, dafür mit sichtbarem Winkel beim Anfügen des Korridors. Unten: Mit geschlossenem Rundweg und angrenzendem Korridor im 90°Winkel.

Newman und Ho haben in [25] einen Weg aufgezeigt, aus Bildern Merkmale zu extrahieren, die sich besonders aus ihrer Umgebung hervorheben und dazu robust identifiziert werden können, auch wenn sie aus unterschiedlichen Richtungen betrachtet werden. Sie messen die "Auffälligkeit" einer Bildregion mittels ihrer lokalen Bildkomplexität, ausgedrückt durch ihre Entropie, gewichtet mit einem Maß der Selbstähnlichkeit auf verschiedenen Skalen. Dazu wird für jede Region ihre Entropie auf diesen Skalen berechnet, worauf ein starker Anstieg(peak) der Entropie bei bestimmten Skalen in einem hohen Gewicht resultiert.
Die Vorarbeit zu diesem Vorgehen haben auch schon [11] und [16] geleistet. Um Merkmale zu finden, die eine Unabhängigkeit von der Blickrichtung haben, wurde ein Detektor implementiert, der MSERs("maximally stable extremal regions") [22] findet. MSERs sind zusammenhängende Regionen im Bild, innerhalb derer keine Sprünge der Farbwerte vorkommen. Aus den Bildregionen, die beide Eigenschaften erfüllen, also ein bestimmtes Maß an Auffälligkeit aufweisen und unabhängig von der Blickrichtung sind, werden SIFT-Deskriptoren extrahiert und in einer Datenbank abgelegt. Bei der Aufnahme eines neuen Bildes sei die Anzahl der daraus extrahierten Deskrip-

toren m, abgespeichert in einem Deskriptorvektor V_q. Der Vergleich mit der Datenbank läuft dann folgendermassen ab: Sei n die Zahl der zu einem Bild in einem Vektor V_c abgespeicherten Deskriptoren. Eine m x n-Adjazenzmatrix wird dann erstellt, wobei der Eintrag (i,j) in der Matrix belegt wird mit der L_2-Norm von $V_q(i) - V_c(j)$. Für jeden Eintrag der Adjazenzmatrix überhalb eines gesetzten Schwellwertes erhält V_c einen Punkt. Diese Punktzahlen geben Hinweis darauf, wie wahrscheinlich es ist, dass die Bilder beider Vektoren den gleichen Ort im Raum abbilden. Unter allen Vektoren in der Datenbank werden deshalb die mit den höchsten Punktzahlen ausgewählt und deren Bilder auf mögliche Schleifenschlüsse mit dem gerade aufgenommenen Bild geprüft.
Stachniss, Hähnel und Burgard beschreiben in [33] die Idee eines "aktiven" Loop-Closing-Verfahrens. Aktiv bedeutet hier, dass das Verfahren selbstständig in die Roboterbewegung eingreifen kann, um ihn an Stellen zu führen, wo mögliche Schleifenschlüsse wahrscheinlich sind. Um festzustellen, wann ein möglicher Schleifenschluss naheliegt, werden zwei Karten simultan gepflegt: Eine Belegtheitsgitterkarte und ein topologischer Graph. Indiz für die Möglichkeit eines Schleifenschlusses sind dann zwei Posen in der Belegtheitsgitterkarte, die dicht beieinander sind, während gleichzeitig der kürzeste Pfad von einem zum anderen im topologischen Graphen lang ist. Die Bedeutung von "lang" und "kurz" wird anhand zweier Schwellwerte festgelegt. Wenn die aktuelle Posenunsicherheit einen bestimmten Wert überschreitet und ein solcher Fall eintritt, wird der Befehl an den Roboter gegeben, beide Orte anzufahren um nach Möglichkeit die Schleife zu schliessen und die Posenunsicherheit damit deutlich zu verringern. Befindet sie sich unterhalb dieses Schwellwertes, so wird stattdessen eine Explorationsstrategie ausgeführt, die versucht, möglichst schnell das Gebiet zu erschliessen, also an zuvor noch nicht besuchte Orte zu fahren.

Kapitel 3

Grundlagen

3.1 Iterative Closest Points

Der Iterative Closest Points Algorithmus(ICP) ist ein iteratives Verfahren, um die wahrscheinlichste Starrkörpertransformation zu finden, die zwei gegebene Punktwolken ineinander überführt [37]. Da Umgebungsinformation oft in Form von Punktwolken vorliegt(z.B. beim Einsatz eines Laserscanners), ist es, häufig in modifizierter Form, eines der Standardverfahren der Robotik, wenn es darum geht räumliche Bewegungen zu schätzen.

Problemstellung: Gegeben sind zwei Punktwolken $X = \{x_1, ..., x_n\}$ und $Y = \{y_1, ..., y_n\}$, je mit Kardinalität n. Gesucht sind eine Rotationsmatrix $\mathrm{R} \in \mathbb{R}^{3x3}$ und ein Translationsvektor $\mathrm{t} \in \mathbb{R}^2$ im zweidimensionalen Fall, bzw. $\mathrm{R} \in \mathbb{R}^{4x4}$ und $\mathrm{t} \in \mathbb{R}^3$ für den dreidimensionalen Fall. R und t sollen dabei die Summe der Quadrate der Abstände korrespondierender Punkte x_i und y_i minimieren.

$$\frac{1}{n}\sum_{i=1}^{n} |x_i - Ry_i - t|^2 \tag{3.1}$$

soll minimal werden. Es gibt verschiedene Methoden um Korrespondenzen zwischen Punkten aus X und Punkten aus Y zuzuweisen. Hierauf wird später kurz eingegangen. Es sei nur Sorge getragen, dass jeder Punkt aus X den gleichen Index trägt wie seine Korrespondenz in Y, damit Gleichung (3.1) stimmt.

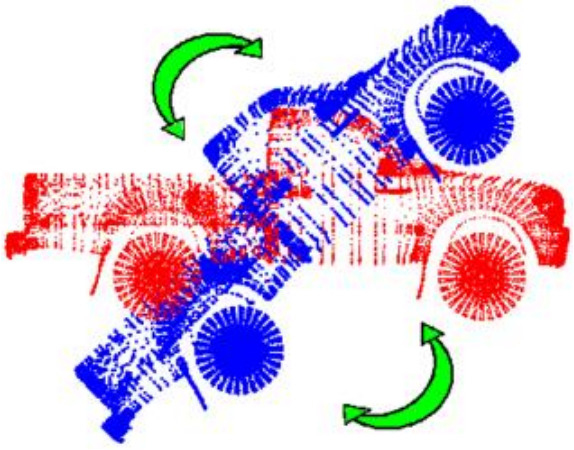


Abbildung 3.1: Welche Transformation überführt die blaue Punktwolke in die rote?

Um die Punktwolken in den Koordinatenursprung zu verschieben, werden die Schwerpunkte von X und Y bestimmt

$$\begin{aligned} \mu_x &= \frac{1}{n}\sum_{i=1}^{n} x_i \text{ bzw.} \\ \mu_y &= \frac{1}{n}\sum_{i=1}^{n} y_i, \end{aligned} \tag{3.2}$$

und von den Punkten abgezogen

$$\begin{aligned} X' &= \{x_i'\} = \{x_i - \mu_x\} \text{ bzw.} \\ Y' &= \{y_i'\} = \{y_i - \mu_y\}. \end{aligned} \tag{3.3}$$

Aus X' und Y' wird nun eine Matrix W folgendermassen erzeugt:

$$W = \sum_{i=1}^{n} x_i' y_i'^T. \tag{3.4}$$

Anwenden einer Singulärwertzerlegung auf W erzeugt eine Darstellung durch drei Matrizen

$$W = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^T (\text{hier: } x_i, y_i \in \mathbb{R}^3 \text{ ; } x_i, y_i \in \mathbb{R}^2 \text{ analog}), \tag{3.5}$$

wobei $\sigma_1 \geq \sigma_2 \geq \sigma_3$ die Singulärwerte von W sind. Nun können R und t bestimmt werden als

$$\begin{aligned} R &= UV^T \\ t &= \mu_x - R\mu_y. \end{aligned} \tag{3.6}$$

Besitzt W keinen Rangdefekt, so sind diese Werte eindeutig. Nun können R und t auf Y angewendet werden. Im Allgemeinen nähern sich die Punktwolken dadurch einander an, was den Fehler aus Gleichung(3.1) verkleinert. Damit endet dieser Iterationsschritt. Anhand von Kriterien wie z.B. der Größe des Fehlers, der Änderungsrate des Fehlers(Differenz zwischen Fehler im letzten Schritt und Fehler in diesem Schritt) oder Festlegung einer Iterationsgrenze kann entschieden werden, ob man dieses Ergebnis behalten möchte, oder ob man die Iteration mit dem transformierten Y wiederholt. Dieses Verfahren konvergiert gegen ein lokales Fehlerminimum, welches nicht notwendigerweise identisch mit dem globalen Minimum ist. Ist die gesuchte Transformation "klein genug" wird in der Regel allerdings eine schnelle Konvergenz gegen das globale Minimum erreicht.

Die Art, Korrespondenzen zwischen X und Y zu bestimmen, hat großen Einfluß auf das Konvergenzverhalten und damit auch auf die Geschwindigkeit des Verfahrens. Wären die korrekten Korrespondenzen vorab bekannt, so wäre die Lösung fehlerfrei in einer Iteration zu berechnen. Da dies im Allgemeinen nicht der Fall ist, muss eine Methode gewählt werden um Korrespondenzen festzulegen. Häufig benutzt werden z.B.:
- nächster Nachbar: Jedem Punkt aus X wird der Punkt aus Y zugewiesen, der noch nicht zugeordnet wurde, und unter diesen den kleinsten euklid'schen Abstand hat.
- Normale bilden: Zu einer lokalen Umgebung um jeden Punkt aus X wird die Normale gebildet, und der Punkt aus Y genommen, der den kleinsten Abstand zu dieser Normalen hat. Dies funktioniert bei der Darstellung von glatten Strukturen besser als die Methode des nächsten Nachbarn, wird aber fehleranfällig, wenn die zugrundeliegende Struktur komplex oder stark verrauscht ist.

Die Zuordnung kann verbessert werden, wenn ein Punktpaar nur dann einander zugewiesen wird, wenn sie zueinander passen. So können etwa die Farbwerte, das lokale Krümmungsverhalten oder beliebige andere Merkmale hierfür verglichen werden. Desweiteren kann es nützlich sein, zu prüfen, welche Punktpaare nicht zu den benachbarten Paaren passen(z.B. durch einen deutlich höheren Punktabstand) und diese als falsche Zuordnungen zu identifizieren. Diese Variante ist auch als Trimmed ICP(TrICP) bekannt. Eine weitere gängige Modifikation ist, das Verfahren von der Bedingung $|X| = |Y|$ zu befreien.

3.1.1 Singulärwertzerlegung

Die Singulärwertzerlegung ist ein mathematisches Verfahren, welches eine Matrix $A \in \mathbb{R}^{mxn}$ vom Rang r in das Produkt $U\Sigma V^T$ zerlegt, wobei $U \in \mathbb{R}^{mxm}$, $V \in \mathbb{R}^{nxn}$ und $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \ddots & 0 & \dots & 0 \\ 0 & 0 & \sigma_r & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & 0 \end{bmatrix}$.
$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ sind die Singulärwerte von A. U und V besitzen die Eigenschaft einer Orthonormalmatrix, d.h. alle Spaltenvektoren von U(V) sind einsnormiert, orthogonal und spannen den $\mathbb{R}^m(\mathbb{R}^n)$ auf.
Die Singulärwertzerlegung findet Anwendung in zahlreichen Gebieten. Dazu zählen die Bildkompression, das Information Retrieval, die Hauptkomponentenanalyse, der ICP-Algorithmus(s.o.) und viele mehr. Weiterhin bietet sie die Möglichkeit, zu einer nicht invertierbaren Matrix eine Pseudoinverse zu berechnen, also eine Matrix, deren Inverse die Originalmatrix möglichst gut approximiert.

Die Zerlegung:
Gegeben: $A \in \mathbb{R}^{mxn}$; Gesucht: U, V, Σ.
Setze B := $A^T A$. $B \in \mathbb{R}^{nxn}$ ist symmetrisch.
MAn bilde nun das charakteristische Polynom von $B = det(B - \lambda I)$ und bestimme seine r Nullstellen, was die Eigenwerte von B sind. Dann ist für jeden Eigenwert λ_i das Gleichungssystem

$$(B - \lambda_i I) * v_i = \mathbf{0}, \tag{3.7}$$

zu lösen, wobei $v_i \in \mathbb{R}^n$ den zugehörigen Eigenvektor ergibt, und $\mathbf{0}$ der n-dimensionale Nullvektor ist. $V := [v_1 v_2 \dots v_n]$ ist eine Orthonormalbasis des $\mathbb{R}^n$. r Vektoren $u_1, ..., u_r$ mit

$$u_i := \frac{1}{\sqrt{\lambda_i}} A v_i. \tag{3.8}$$

müssen erzeugt werden und anschliessend die Vektoren $u_{r+1}, \dots, u_m$ so ergänzt werden, dass $U := [u_1, \dots, u_m]$ eine Orthonormalbasis des $\mathbb{R}^m$ bildet. Die ersten r Diagonaleinträge $\sigma_{i,i}$ von $\Sigma \in \mathbb{R}^{mxn}$ sind die Quadratwurzeln der λ_i. Diese sind reell, da $\forall 1 \leq i \leq r : \lambda_i > 0$. Die restlichen Einträge der Matrix sind 0.[23]

3.2 Das Registrierungsverfahren

Das Registrierungsverfahren basiert auf der Verwendung von sog. "multiresolution Surfel maps" [35]. Diese verwenden Octrees um den Raum dynamisch auf verschiedenen Auflösungen repräsentieren zu können. Jeder Knoten des zugrundeliegenden Octrees repräsentiert eine kubische Raumeinheit(Voxel). Die Belegung einer MRSM erfolgt durch die Übergabe farbiger Punktwolken. Deren Information wird ausgewertet, und darauf basierend erfolgt die Strukturierung des Octrees, sowie die Belegung der Knoten. Die Tiefe eines Knotens steht in direktem Zusammenhang mit der durch ihn repräsentierten Kartenauflösung. Der Erste der folgenden drei Abschnitte behandelt das Erstellen einer MRSM auf Basis von Punktwolken, also die Umwandlung der Repräsentation. Die Registrierung zweier MRSMs aufeinander gliedert sich im Anschluss in zwei Abschnitte: Im ersten Teil werden korrespondierende Surfel gesucht, die in beiden MRCSMs vorkommen und dasselbe Oberflächenstück repräsentieren. Für alle diese Paare wird im zweiten Teil eine gemeinsame Transformation gefunden, die maximale Zuordnungswahrscheinlichkeit besitzt.

3.2.1 Repräsentation durch multiresolution Surfel maps

Multiresolution color Surfel maps repräsentieren farbige Oberflächenstücke(engl. "surface elements", kurz "Surfel") anhand der gemeinsamen Verteilung von Farbe und Form auf unterschiedlichen Auflösungen. Die Belegung einer MRSM erfolgt durch die Übergabe einer oder mehreren farbigen Punktwolken. Jeder Punkt dieser Punktwolken ist ein 6D-Vektor, bestehend aus einer Raumposition(X,Y,Z) und eines RGB-Farbwertes. Der dreidimensionale Raum wird mittels eines Octrees repräsentiert, dessen Knoten würfelförmige Volumenelemente(engl. "volume elements", kurz "Voxel") darstellen. Die Punkte, die in einen Voxel hineinfallen, werden normalverteilt modelliert. Jeder Knoten dieses Octrees erhält zunächst statistische Informationen S und S^2, d.h. aus allen Punkten p der Punktmenge P in einem Voxel werden

$$\begin{aligned} S(P) &:= \sum_{p \in P} p \text{ und} \\ S^2(P) &:= \sum_{p \in P} pp^T \text{ berechnet.} \end{aligned} \tag{3.9}$$

Durch Bildung von Mittelwert $\mu(P) = \frac{1}{|P|} S(P)$ und Kovarianzmatrix $\Sigma = \frac{1}{|P|} S^2(P) - \mu\mu^T$ wird ein normalverteiltes Modell erstellt. Seien zwei Punktwolken P^A und P^B gegeben, und setzt man $\delta := |P^B| S(P^A) - |P^A| S(P^B)$,

dann sind die statistischen Werte der Vereinigung von P^A und P^B definiert als

$$\begin{aligned} S(P^{A\cup B}) &:= S(P^A) + S(P^B), \text{ und} \\ S^2(P^{A\cup B}) &:= S^2(P^A) + S^2(P^B) + \frac{\delta\delta^T}{|P^A| \cdot |P^B| \cdot (|P^A| + |P^B|)}. \end{aligned} \tag{3.10}$$

Jedes Surfel fasst zwischen 10 und 10000 Punkten zusammen. Fallen in ein Surfel weniger als 10 Punkte, so wird es verworfen. Umgekehrt werden alle Punkte ignoriert, die in ein Surfel fallen, was bereits 10000 andere Punkte enthält. Die errechneten Kovarianzen können auf Grund von Diskretisierungseffekten des RGBD-Sensors numerisch instabil sein. Dies wird durch den Vergleich ihrer Determinante mit einem kleinen Schwellwert überprüft.

Um die Helligkeit vom Farbwert zu trennen wird eine Abwandlung des HSL(hue = Farbton, saturation = Sättigung, lightness = relative Helligkeit)-Farbraumes definiert - der $L\alpha\beta$-Raum:

$$L := \frac{1}{2}(max\{R, G, B\} + min\{R, G, B\}) \tag{3.11}$$

$$\alpha := R - \frac{1}{2}G - \frac{1}{2}B \tag{3.12}$$

$$\beta := \frac{\sqrt{3}}{2}(G - B) \tag{3.13}$$

H und S werden durch α und β ersetzt. Die Berechnung von L ist identisch mit der aus dem HSL-Raum.

3.2.2 Form-Textur-Deskriptoren

Für jeden Knoten im Octree werden Deskriptoren für Form und Textur, bezüglich der Nachbarschaft des Surfels erzeugt. Zu jedem Deskriptor werden zunächst drei Histogramme folgendermassen erzeugt:
Der Octree O habe die Tiefe n. O(i) sei die i-te Auflösungsstufe und $s_{i,j}$ das j-te Surfel in O(i). $\forall 1 \leq i \leq n : \forall 1 \leq j \leq |O(i)|$: Vergleiche $s_{i,j}$ mit allen direkt benachbarten Surfeln $s_{i,k}$, $k \neq j$ folgendermassen: Erstelle das erste Histogramm von $s_{i,j}$ aus den Winkeln zwischen den Normalen von $s_{i,j}$ und denen von $s_{i,k}$. Das zweite Histogramm wird aus den Winkeln zwischen der Normalen von $s_{i,j}$ und der Verlängerung von $\Delta\mu := \mu_{s_{i,j}} - \mu_{s_{i,k}}$ gebildet. Das dritte entsteht Aus den Winkeln zwischen den $s_{i,k}$ und $\Delta\mu$. Dabei wird jeder Winkel vor dem Einfügen mit der Kardinalität der Punkte in $s_{i,k}$ gewichtet. Um Diskretisierungseffekten vorzubeugen, wird im Anschluss auf

jedes Histogramm die Summe aller benachbarten Histogramme, skaliert mit dem Faktor 0.1, hinzuaddiert, und das Ergebnis zum Normalisieren durch die Gesamtzahl der Punkte geteilt.
Auf ähnliche Weise werden zusätzlich weitere lokale Histogramme für Kontraste im L-, α- und β-Wert extrahiert. Dabei erfolgt für jedes benachbarte Surfelpaar eine Einteilung in positiv, negativ und unerheblich. Zur Laufzeitverbesserung werden die Pointer auf Nachbarsurfel vorberechnet, und in den Knoten abgelegt. Zudem muss man die Statistiken nicht für jeden Punkt einzeln berechnen und zur Struktur hinzufügen, sondern kann ausnutzen, dass im Bild benachbarte Pixel mit hoher Wahrscheinlichkeit zu Punkten desselben Surfels gehören, sofern dort kein Tiefensprung stattfindet. Deswegen wird das Bild durchsucht, um statistische Informationen benachbarter Regionen zu sammeln, die im selben Octree-Knoten zusammengefasst werden. Diese können so für ganze Knoten auf einmal eingefügt werden. Das reduziert die Anzahl der Einfügeoperationen bei einer Auflösung von 640x480 von 307200(jeder Raumpunkt wird einzeln behandelt) auf einige Tausend.
Typisch für RGBD-Kameras ist, dass die Ungenauigkeit in der Position eines erzeugten 3D-Punktes proportional zum Abstand vom Sensor ist. Daher wird die maximale Octree Auflösung an einem Pixel auf das Quadrat dieses Abstandes gesetzt. Somit reduziert sich der Speicherbedarf des Octrees, und seine Blätter fassen repräsentative Oberflächenstücke zusammen. Gesondert behandelt werden müssen Bildgrenzen, sowie virtuelle Grenzen, die durch Tiefensprünge entstehen. Dort wird die zu Grunde liegende Oberflächenstruktur nur Teilweise erfasst, was zu einer Verzerrung der ermittelten Statistiken S(P) und $S^2(P)$ und damit zu einem suboptimalen Ergebnis führt. Daher wird das Bild nach solchen Grenzen durchsucht, um betroffene Knoten aus künftigen Berechnungen auszuschliessen.

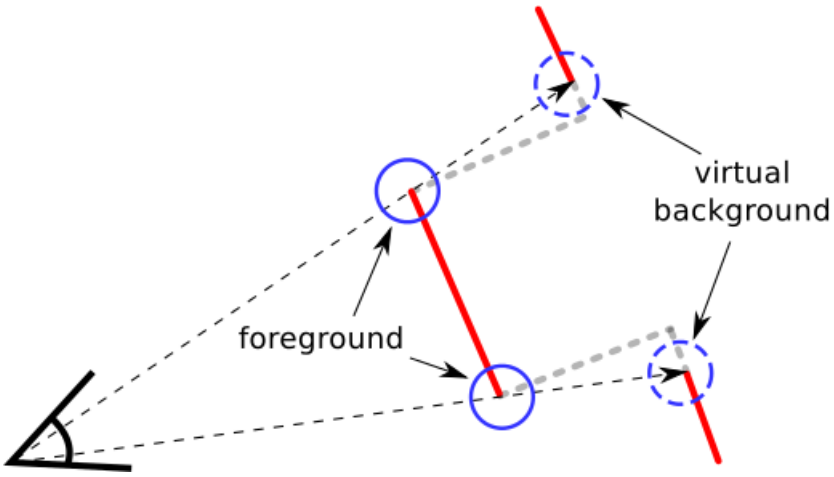


Abbildung 3.2: Virtuelle Bildgrenzen(Tiefensprünge): Teile des Hintergrundes sind von Objekten im Vordergrund verdeckt.

3.2.3 Das Zuordnen von Surfeln

Beginnend bei der feinsten Auflösung, wird durch alle Knoten derselben Auflösungsstufe iteriert, um Korrespondenzen zwischen Surfeln aus verschiedenen Blickrichtungen zu finden. Knoten, die mindestens einen bereits zugeordneten Sohn besitzen, werden nicht mehr zugeordnet. Dies spart zum einen redundante Berechnungen, zum anderen finden damit die Zuordnungen in den jeweils kleinstmöglichen Auflösungen statt. Da sich die Suche nach Surfelpaaren und die Posenschätzung mehrfach abwechseln, werden die Zuordnungen abgespeichert, und nur für die bisher nicht zugeordneten Surfel nach Partnern gesucht. In der Ziel-MRSM wird nach Surfeln gesucht, die die doppelte Kantenlänge des Anfragesurfels haben. In nachfolgenden Iterationen werden dadurch Zuordnungsfehler zunächst auf gröberen Auflösungen korrigiert, die sukzessiv kleiner werden. Wenn bereits eine Assoziation aus einer früheren Iteration besteht, so findet die Zuordnung erneut unter den 26 direkten Nachbarn des Zielsurfels statt. Da für jedes Surfel die Verweise auf alle seine Nachbarn bereits vorberechnet wurden, erfolgt diese Korrektur in konstanter Zeit. Zuordnungen werden nur akzeptiert, wenn die jeweiligen Deskriptoren zueinander passen. Dies wird festgelegt anhand der Summe der euklid'schen Abstände zwischen den Deskriptoren für Form, L, α, β. Nur wenn sie einen Wert von 0,1 nicht überschreitet ist die Zuordnung gültig. Innerhalb einer Auflösungsstufe werden alle Surfel unabhängig voneinander aktualisiert, so dass sich hier eine Möglichkeit zur Parallelisierung bietet.

3.2.4 Die Transformationsschätzung

Wir bezeichnen im Folgenden die Quell-MRSM mit M_q, die Ziel-MRSM mit M_z und eine Raumpose mit $x = (q, t)^T$, wobei q die Rotation durch ein Einheitsquaternion darstellt, und t den Translationsvektor. T(x) sei die homogene Matrix der gesamten Transformation von x, und R(x) die Rotationsmatrix. Weiterhin ist A die Menge der zugeordneten Surfels zwischen M_q und M_z, $s_{q,i} = (\mu_{q,i}, \Sigma_{q,i})$ ist das i-te Surfel in M_q und $s_{z,j} = (\mu_{z,j}, \Sigma_{z,j})$ ist das j-te Surfel in M_z.
Ziel ist, die Pose x zu finden, für die $p(M_q|x, M_z)$, als Wahrscheinlichkeit, M_z an der Pose x in M_q wiederzufinden maximal wird. Die Wahrscheinlichkeit der gesamten Zuordnung ergibt sich aus den Wahrscheinlichkeiten der Zuordnungen einzelner Surfel.

$$p(M_q|x, M_z) = \prod_{(i,j)\in A} p(s_{q,i}|x, s_{z,j}) \tag{3.14}$$

Die Zuordnungswahrscheinlichkeit der Surfel ist ihr Abstand, bezogen auf ihre Normalverteilungen. Definiert man

$$\begin{aligned} d_{i,j}(x) &:= \mu_{z,j} - T(x)\mu_{q,i}, \text{ und} \\ \Sigma_{i,j}(x) &:= \Sigma_{z,j} + R(x)\Sigma_{q,i}R(x)^T, \end{aligned} \tag{3.15}$$

dann kann die Zuordnungswahrscheinlichkeit ausgedrückt werden durch

$$p(s_{q,i}|x, s_{z,j}) = N(d_{i,j}(x); 0, \Sigma_{i,j}(x)). \tag{3.16}$$

Wegen unterschiedlicher Blickrichtungen der RGBD-Bilder wird auch deren Inhalt unterschiedlich diskretisiert. Daraus resultierende Ungenauigkeiten werden durch eine trilineare Interpolation unter benachbarten Surfeln ausgeglichen.
Optimiert wird der Logarithmus der Zuordnungswahrscheinlichkeit

$$L(x) = \sum_{((i,j)\in A)} log(|\Sigma_{(i,j)}(x)|) + d^T_{(i,j)}(x)\Sigma^{-1}_{(i,j)}(x)d_{(i,j)}(x) \tag{3.17}$$

für die Pose x in zwei Schritten: Zuerst wird die Levenberg-Marquardt-Methode angewandt, um einen Näherungswert zu bestimmen. Mit diesem wird dann die Newton-Methode initialisiert, die ein exakteres Ergebnis liefert. Sei $e(x) := (d_{(i,j)}(x))_{(i,j)\in A}$ der Fehlervektor der Surfelresiduen und W eine Gewichtsmatrix. Die Levenberg-Marquardt-Methode löst gewichtete nichtlineare Optimierungsprobleme der Art $e(x)^T W e(x)$. Der Fehlervektor ergibt sich aus den Differenzen der Deskriptoren korrespondierender Surfel. Der Einfluss

der Posen auf die Kovarianzen wird in diesem Fall vernachlässigt, und wenn $(i, j) \in A$ ein Paar aus der Menge A der zugeordneten Surfel ist, dann ergibt sich eine konstante, blockdiagonale Gewichtsmatrix $W = diag(w_{(i,j)}\Sigma^{-1}_{(i,j)}(x))$. Hierbei ist $w_{(i,j)} = 0,1 - d_{(i,j)}(x)$ Sei J eine Jakobimatrix, entstanden durch Zusammensetzung der Jakobimatrizen der geschätzten Transformationen aller Surfelzuordnungen. Die LM-Optimierung aktualisiert nun die Posen x im Sinne einer gedämpften Gauss-Newton Iteration

$$x' := x + (J^T W J + \lambda I)^{-1} J^T W e(x). \tag{3.18}$$

Während dieser Optimierung findet keine trilineare Interpolation zwischen den Surfeln statt, und neue Zuordnungen finden nur dann statt, wenn die Posenänderung pro Schritt unter einen Schwellwert gerutscht ist. Ändert sich die Pose auch anschließend nicht, oder ist eine Obergrenze für die Iterationszahl erreicht, so stoppt der Algorithmus. Das Ergebnis wird als Eingabe für Newtons Methode(Glg. 3.17) verwendet, bei der die trilineare Interpolation und die Surfel Zuordnung in jedem Schritt stattfindet. Newtons Methode benötigt Ableitungen zweiter Ordnung, die Dank der simplen Struktur von Gleichung (3.17) analytisch berechnet werden können. Das Berechnen der ersten und zweiten Ableitungen ist gut parallelisierbar, was in einem deutlichen Geschwindigkeitszuwachs auf Mehrkern-Prozessoren resultiert.
Die Normalisierung der Quaternionen in der Posendarstellung bedarf gesonderter Behandlung: Optimiert wird nur der Imaginärteil des Quaternions, während sich der Realteil anschließend aus der Normalisierungsbedingung und seinem Vorzeichen vor der Optimierung rekonstruieren lässt. Dieses Vorgehen allein ist fehlerhaft für Abweichungen $\geq 180°$. Um diesen Umstand zu beheben und Gültigkeit für beliebige Abweichungen zu erzielen, wird die Posenschätzung x aus einem konstanten Teil x' und einer Änderung Δx als $T(x) = T(\Delta x)T(x')$ erstellt. In jeder Iteration von LM oder der Newton Methode wird x' = x gesetzt und die Änderung Δx optimiert. Konvergenz wird in der Regel nach 10-20 Iterationen des LM-Verfahrens, und 5 Iterationen des Newton-Verfahrens erzielt.
Die Kovarianz im Sinne der relativen Posenunsicherheit zwischen einem Surfelpaar s wird durch eine geschlossene Form [2] bestimmt. Sei L(x) der Logarithmus der Zuordnungswahrscheinlichkeit aus Gleichung (3.17), s sei die Assoziation zweier Surfel, und $\Sigma(s)$ ihre Kovarianz. Dann ist

$$\Sigma(x) \approx (\frac{\delta^2 L(x)}{\delta x^2})^{-1} \frac{\delta^2 L(x)}{\delta s \delta x} \Sigma(s) (\frac{\delta^2 L(x)}{\delta s \delta x})^T (\frac{\delta^2 L(x)}{\delta x^2})^{-1}. \tag{3.19}$$

3.2.5 Multi-Frame Registrierung

Eine Erweiterung des Registrierungsverfahrens bietet die Möglichkeit, mehrere Quell-MRSMs $M_{q,i}$ gegen mehrere Ziel-MRSMs $M_{z,j}$ gemeinsam zu registrieren. Dabei besteht die Möglichkeit, Vorwissen über unsichere Bedingungen in der gegenseitigen Lage der Quell-MRSMs anzugeben. Seien $x_{z,j}$ die Posen der Ziel-MRSMs in einem gemeinsamen Koordinatensystem. Diese Posen werden von der Registrierung als unveränderlich angesehen. Geschätzt werden die relativen Posen der $M_{q,i}$ gegen die $M_{z,j}$ unter Berücksichtigung der vorab übergebenen Lagebedingungen. Eine Bedingung zwischen zwei Quell-MRSMs $M_{q,i}$ und $M_{q,j}$ besteht aus Mittelwert x_i^j und Kovarianzmatrix Σ_i^j.

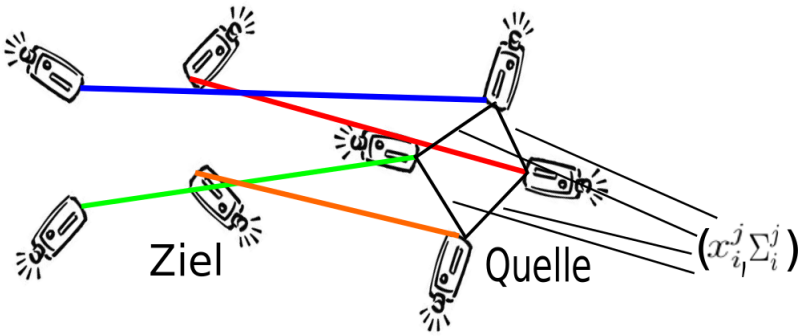


Abbildung 3.3: Mehrere Maps(Quelle) mit Bedingungen (x_i^j, Σ_i^j) werden auf mehrere Maps(Ziel) registriert.

Dadurch entsteht ein Optimierungsproblem bezüglich der resultierenden Posen der $M_{q,i}$, welches mit der Levenberg-Marquardt-Methode(siehe Kap. 3.3 ("G^2O")) gelöst wird. Bezeichne T(x) die Umwandlung einer Pose zu ihrer entsprechenden Transformationsmatrix und sei $\widehat{x}_i^j$ die geschätzte relative Pose von $M_{q,i}$ nach $M_{q,j}$. Diese ergibt sich aus den Schätzungen x_i und x_j und der Posendifferenz $x_{z,k}^{z,l} := x(T(x_{z,j})^{-1}T(x_{z,i}))$ der Ziel-MRSMs als

$$\widehat{x}_i^j := x(T(x_j)^{-1}T(x_{z,i}^{z,j})T(x_i)). \tag{3.20}$$

Bezeichne ferner $\Delta(x_i^j, \widehat{x}_i^j) := x(T(x_i^j)^{-1}T(\widehat{x}_i^j))$ die Differenz zwischen geschätzter Pose und dem übergebenen Mittelwert. Dann ist der Fehler

$$E(x) = (\sum_i \sum_{a \in A} d_a^T(x_i)\Sigma_a^{-1}(x_i)d_a(x_i)) + (\sum_{i,j}(\Delta(x_i^j, \widehat{x}_i^j))^T(\Sigma_i^j)^{-1}\Delta(x_i^j, \widehat{x}_i^j)). \tag{3.21}$$

Wie im Falle der vorangegangenen Einzelregistrierung werden die Residuen der Surfelzuordnungen zwischen allen Karten zusammen mit den Residuen

der Posenbedingungen in einen gemeinsamen Residuenvektor $e(x)$ gesteckt. Die relativen Posen der N registrierten MRSMs werden in einem Zustandsvektor $x := (x_1, ..., x_N)^T$ untereinander geschrieben. Die Gewichtsmatrix W ist wieder blockdiagonal, wobei ihre Diagonaleinträge die Kovarianzen der Residuen sind. Die Jakobimatrizen der Surfelresiduen $J_a(x)$ und der Residuen der gegenseitigen Lagebedingungen $J_{i,j}(x)$ werden in einer Jakobimatrix J(x) zusammengefügt. Analog dazu, dass ein Knoten in einem Graphen mit mehreren Kanten verbunden sein kann, kann jede relative Pose x_i unter dem Einfluss mehrerer Lagebedingungen stehen. Jede Bedingung wirkt sich im Allgemeinen durch die partiellen Ableitungen von $\Delta(x_i^j, \widehat{x}_i^j)$ auf beide betreffenden Posen x_i und x_j aus.

3.2.6 Ergebnisse

Die Registrierungsmethode wurde in ein RGBD-SLAM-Verfahren mit loopclosure-Erkennung und Posengraphoptimierung mittels G^2O(siehe Kap. 3.3 ("G^2O")) eingebaut. Dieses wurde anhand öffentlich zugänglicher RGBD-Datensätze [34] getestet, und die Ergebnisse an Hand zweier, dort vorgeschlagener Metriken(ATE = absolute trajectory error; RPE = relative pose error) bewertet. Zusätzlich wurden weitere Datensätze erstellt, die 360° Trajektorien um ein fokussiertes Objekt enthalten. Die Objekte sind eine Kiste, ein kleiner Humanoid und ein Bürostuhl. Jedes Objekt wurde einmal mit langsamer, mittlerer und mit schneller Kamerabewegung abgefilmt, wodurch es sich um neun weitere Datensätze handelt. Die Aufnahmefrequenz lag bei 30Hz und die Auflösung betrug 640x480(VGA). Die minimale Auflösung der MRSMs wurde auf 0.0125m gesetzt. Die Rechenzeiten wurden auf einer Intel Core i7 3610QM @2300MHz QuadCore CPU ermittelt. Die folgende Tabelle zeigt einen Vergleich zwischen diesem und anderen populären Verfahren hinsichtlich des RPE(relative pose error) auf den Datensätzen aus Freiburg.

Datensatz	diese Methode	warp (OpenCV)	GICP	3D-NDT	fovis
fr1 360	5,1(46,4)	5,9(75,4)	18,8(88,3)	7,8(140,8)	7,1(43,1)
fr1 desk	4,4(20,1)	5,8(131,8)	10,2(54,9)	7,9(26,6)	6,3(34,2)
fr1 desk2	4,5(24,1)	6,2(147,4)	10,4(261,2)	8,2(46,3)	6,6(49,9)
fr1 floor	4,9(355,9)	2,1(3167)	5,0(193,6)	6,3(1021)	2,6(412,4)
fr1 plant	3,5(27,7)	4,2(300,8)	16,1(831,4)	7,4(62,3)	4,6(61,6)
fr1 room	3,5(33,3)	4,6(167,8)	10,2(212,6)	6,1(51,2)	5,4(55,1)
fr1 rpy	3,0(24,8)	5,1(41,8)	10,4(636,6)	6,8(41,9)	5,4(38,7)
fr1 teddy	4,2(77,4)	6,1(381,1)	21,3(356,6)	8,8(126,6)	7,1(82,4)
fr1 xyz	2,6(9,8)	4,1(18,1)	3,9(42,0)	5,2(36,7)	4,6(25,8)
fr2 desk	2,1(15,1)	2,1(14,1)	6,7(64,4)	4,3(31,7)	2,5(15,5)
fr2 large no loop	21,8(167,4)	20,5(76000)	21,3(1289)	32,1(2512)	11,0(173,0)
fr2 rpy	1,6(29,9)	1,7(189,5)	1,3(28,7)	4,2(55,1)	1,7(11,0)
fr2 xyz	1,4(33,1)	2,0(8,8)	1,7(26,8)	4,0(18,0)	1,9(9,9)
fr2 long office household	2,6(16,9)	3,2(34,0)	7,8(209,1)	4,2(40,2)	3,7(35,6)
fr3 nostruct. notext. far	9,7(48,4)	40,4(6000)	8,6(66,1)	13,8(77,8)	11,3(108,4)
fr3 nostruct. notext. near	15,2(56,9)	28,2(3200)	12,5(182,1)	17,1(144,6)	11,2(79,3)
fr3 nostruct. text. far	18,5(57,9)	19,2(1230)	10,9(58,6)	18,6(74,6)	20,8(101,5)
fr3 nostruct. text. near	11,5(52,4)	7,0(100,5)	8,9(67,1)	10,6(80,4)	7,3(41,6)
fr3 struct. notext. far	2,2(16,6)	8,6(2579)	4,5(23,2)	2,9(19,2)	9,1(62,4)
fr3 struct. notext. near	2,1(13,5)	8,6(1108)	2,9(12,2)	2,4(44,5)	9,3(86,9)
fr3 struct. text. far	5,5(20,7)	8,1(39,0)	7,1(23,3)	5,4(20,1)	8,8(45,2)
fr3 struct. text. near	3,2(14,7)	5,9(34,8)	5,6(34,2)	5,5(82,3)	6,5(38,2)

Tabelle 3.1: Vergleich der Verfahren hinsichtlich Median(Maximum) des RPE(relative pose error) in mm auf den Freiburg Benchmarkdatensätzen

Der Vergleich der Laufzeiten unter den verschiedenen Verfahren folgt in der nächsten Tabelle.

Tabelle 3.2: Vergleich der Verfahren hinsichtlich Durchschnitt(Standardabweichung) der Laufzeit pro Registrierungsvorgang in Millisekunden

Datensatz	diese Methode	warp(OpenCV)	GICP	3D-NDT	fovis
fr1 desk	75,15(9,66)	108,64(18,41)	4015,4(1891,6)	414,87(255,57)	15,98(7,0)
fr2 desk	61,47(7,68)	99,29(10,61)	3147,3(852,98)	892,59(253,54)	12,98(2,27)

Tests auf den eigens erstellten Datensätzen(Humanoid, Kiste und Bürostuhl durch langsame, moderate und schnelle Kamerabewegung) lieferten Ergebnisse, deren ATE(absolute trajectory error) und Laufzeiten ebenfalls im Folgenden tabellarisch dargestellt sind.

Tabelle 3.3: ATE(absolute trajectory error) in mm, sowie durchschnittliche Laufzeit $\pm$ Standardabweichung in Millisekunden

Datensatz	ATE	Laufzeit
Humanoid(langsam)	19,99	31,39 ± 5,89
Humanoid(mittel)	26,12	32,50 ± 5,19
Humanoid(schnell)	31,64	33,55 ± 6,66
Kiste(langsam)	23,55	39,01 ± 4,89
Kiste(mittel)	37,83	41,15 ± 14,20
Kiste(schnell)	24,05	33,87 ± 9,19
Stuhl(langsam)	22,87	49,09 ± 7,13
Stuhl(mittel)	15,78	49,57 ± 9,93
Stuhl(schnell)	27,67	48,78 ± 10,24

3.3 G^2O

G^2O steht für "general (hyper) Graph optimizer", und bezeichnet eine Programmbibliothek für allgemeine Graphoptimierungsprobleme. Allgemein gesehen ist ein kleinste-Quadrate-Minimierungsproblem von folgender Art: Sei $x = (x_1^T, ..., x_n^T)^T$ ein Vektor, der allgemeine, in Blöcke eingeteilte Eingabeparameter enthält, $x_k = (x_{k_1}, ..., x_{k_q}) \subset x$ eine, durch das k-te Element einer Menge C von Bedingungen festgelegte, Teilmenge der Parameter. μ_k und Σ_k seien Mittelwert und Kovarianzmatrix der k-ten Bedingung, und $e_k(x_k, \mu_k)$ sei eine Fehlerfunktion, die ausdrückt, wie gut die Parameter in x_k die Bedingung μ_k erfüllen. Im Falle totaler Übereinstimmung sei das Ergebnis $= 0$.
Wir vereinfachen im Folgenden die Notation der Fehlerfunktion von $e_k(x_k, z_k)$ zu $e_k(x)$

Nun soll ein x^* gefunden werden, so dass

$$x^* = \min_x(F(x)) \tag{3.22}$$

, wobei

$$F(x) = \sum_{k \in C} \underbrace{e_k(x)^T \Sigma_k e_k(x)}_{F_k} \tag{3.23}$$

$\forall i, j \in \{1, ..., n\}; i \neq j$ müssen $(x_i, e_i(x))$ und $(x_j, e_j(x))$ sich nicht notwendigerweise auf den gleichen Datentyp beziehen. Der Grundgedanke ist, dass man jedes nichtlineare Optimierungsproblem durch einen (hyper)-Graph darstellen kann. Ein Knoten im Graph repräsentiert je einen Parameterblock, und eine (hyper)-Kante steht für eine Bedingung zwischen allen, zu ihr inzidenten, Knoten.

3.3.1 kleinste Quadrate Lösung

Sei $\widehat{x}$ eine Initialbelegung für x. Die Fehlerfunktion wird mittels Taylorexpansion erster Ordnung um $\widehat{x}$ approximiert. Wir bezeichnen die Jakobimatrix für $e_k(x)$ an der Stelle $\widehat{x}$ mit J_k.

$$e_k(\widehat{x} + \Delta x) \simeq e_k + J_k \Delta x \tag{3.24}$$

Substituiert man in den F_k aus Gleichung 3.12 gemäß Gleichung 3.13, kann man wie folgt zusammenfassen:

$$F_k(\widehat{x} + \Delta x) \tag{3.25}$$

$$= e_k(\widehat{x} + \Delta x)^T \Sigma_k e_k(\widehat{x} + \Delta x) \tag{3.26}$$

$$\simeq (e_k(\widehat{x}) + J_k \Delta x)^T \Sigma_k (e_k(\widehat{x}) + J_k \Delta x) \tag{3.27}$$

$$= \underbrace{e_k(\widehat{x})^T \Sigma_k e_k(\widehat{x})}_{c_k} + 2 \underbrace{e_k(\widehat{x})^T \Sigma_k J_k}_{b_k} \Delta x + \Delta x^T \underbrace{J_k^T \Sigma_k J_k}_{H_k} \Delta x \tag{3.28}$$

$$= c_k + 2 b_k \Delta x + \Delta x^T H_k \Delta x \tag{3.29}$$

Schreibt man $\sum_{k \in C} b_k = b$, $\sum_{k \in C} c_k = c$ und $\sum_{k \in C} H_k = H$, erhalten wir für Gleichung 3.12:

$$F(\widehat{x} + \Delta x) = \sum_{k \in C} F_k(\widehat{x} + \Delta x) \tag{3.30}$$

$$\simeq \sum_{k \in C} c_k + 2 b_k \Delta x + \Delta x^T H_k \Delta x \tag{3.31}$$

$$= c + 2 b^T \Delta x + \Delta x^T H \Delta x \tag{3.32}$$

Der Versuch, einen Wert für Δx zu finden, der Gleichung 3.21 minimiert, führt über die Lösung folgenden Gleichungssystems:

$$H \Delta x^* = -b \tag{3.33}$$

H ist die Kovarianzmatrix des gesamten Systems. Sie hat genau dort Einträge $\neq 0$, wo zwei Blöcke durch eine Bedingung verknüpft sind. Daher kann in den meisten Fällen von einer dünnbesetzten Matrix gesprochen werden. Dieses Gleichungssytem kann sehr effizient z.B. über die Methode der dünnbesetzten Colesky-Zerlegung oder preconditioned conjugate gradients(PCG) gelöst werden. Eine Implementation der ersten Vorgehensweise findet sich in öffentlich zugänglichen Softwarepaketen wie CSparse[6] und CHOLMOD[3].

Diese Lösung arbeitet iterativ, und in jedem Iterationsschritt wird der Eingabewert um die neu ermittelte Änderung korrigiert:

$$x^* = \widehat{x} + \Delta x^* \tag{3.34}$$

Das Gauss-Newton-Verfahren verwendet den Linearisierungsschritt(Glg. 3.21), die Lösung des Gleichungssystems(3.22) und den update-Schritt(Glg. 3.23). Der Unterschied zur Levenberg-Marquardt-Methode besteht darin, dass diese nicht (3.22) löst, sondern eine gedämpfte Version dessen

$$(H + \lambda I)\Delta x^* = -b \tag{3.35}$$

λ ist der Dämpfungsfaktor: Je größer das λ, desto kleiner die Änderung in x^*. So kann die Schrittgröße in jedem Schritt an den Fehler aus dem letzten Schritt angepasst werden. Genauere Angaben zu dieser Version sind unter[20] zu finden.
Der bisher beschriebene Ansatz geht davon aus, dass der zu Grunde liegende Parameterraum euklid'sch ist, was für Probleme wie SLAM oder Bündelausgleichung nicht zutrifft. Die Folge wären suboptimale Lösungen. Daher wird der Ansatz später auch auf nichtlineare Parameterräume erweitert.

Jede Kante im Graph trägt einen zusätzlichen Term zum Gleichungssystem bei. Dieser Term hängt von der Jakobimatrix der zugehörigen Fehlerfunktion $e_k(x)$ ab, die wiederum nur von allen Knoten $x_{k_i} \in x_k$ abhängt. Demnach hat die Jakobimatrix folgende Form:

$$J_k = (0, ..., 0, J_{k_1}, ..., J_{k_i}, ..., 0, ..., J_{k_q}, 0, ..., 0) \tag{3.36}$$

Die $J_{k_i} = \frac{\delta e(x_k)}{\delta x_{k_i}}$ sind die Ableitungen der Fehlerfunktion bezüglich aller Knoten die durch die k-te (hyper)-Kante verbunden werden, sowie des Parameterblocks $x_{k_i}\ in x_k$. Die Struktur von Matrix H, konstruiert nach Gleichung(3.17), entspricht der einer Adjazenzmatrix des (hyper)-Graphen, was zur Folge hat dass sie symmetrisch ist.

Wenn man beispielsweise das SLAM-Problem betrachtet, fällt auf, dass der Raum der Translationen noch euklidsch ist. Die Rotationen dagegen spannen aber die SO(2), bzw. SO(3), also die Gruppe der ebenen, bzw. räumlichen Rotationen auf. Wählt man für die Rotationen eine überbestimmte Repräsentation wie z.B. Rotationsmatrizen oder Quaternionen, so führt man einen zusätzlichen Freiheitsgrad in das System ein, was sich in Fehlern im Ergebnis niederschlägt. Wählt man die Minimalrepräsentation, kann es zu Singularitäten kommen. Es existiert also die Notwendigkeit, das oben beschriebene Verfahren auf nicht-euklid'sche Räume zu erweitern. Dazu kann man den zugrundeliegenden Raum auf eine Mannigfaltigkeit abbilden,

indem man einen Operator $\boxplus$ definiert, der dies realisiert, also Veränderungen Δx im euklid'schen Raum auf Veränderungen auf einer Mannigfaltigkeit abbildet, $\Delta x \mapsto x \boxplus \Delta x$. Mehr Details zu diesem Vorgehen findet man unter [14]. Dieser Operator ermöglicht es, die Fehlerfunktion abzuwandeln, wofür vorher festgelegt sei, dass für jede Veränderliche Z, $\tilde{Z}$ ihr Vertreter in Minimalrepräsentation ist.

$$\widehat{e}_k(\tilde{x}_k) \stackrel{def.}{=} e_k(\widehat{x}_k \boxplus \Delta\tilde{x}_k) \tag{3.37}$$

$$= e_k(\widehat{x} \boxplus \Delta\tilde{x}) \simeq e_k(\widehat{x}) + \tilde{J}_k\Delta\tilde{x} \tag{3.38}$$

Wenn man das Beispiel mit den Rotationen fortführt, so ist dann $\Delta\tilde{x} = (\Delta\tilde{t}, \tilde{q})$ ein 6D-Vektor, wobei $\Delta\tilde{t}$ der normale Translationsvektor ist, und $\tilde{q}$ für den Vektorteil des normierten Quaternions steht. Der Operator $\boxplus$ konvertiert also erst $\tilde{q}$ in ein echtes Quaternion q und wendet $(\Delta t(= \Delta\tilde{t}), q)$ als Transformationsupdate an. Die Beschreibungen der Fehlerminimierung ändern sich dadurch nach

$$\tilde{J}_k \underset{\Delta\tilde{x}\to 0}{=} \frac{\delta e_k(\widehat{x} \boxplus \Delta\tilde{x})}{\delta\Delta\tilde{x}} \tag{3.39}$$

$$= (0, ..., 0, \tilde{J}_{k_1}, ..., \tilde{J}_{k_i}, ..., 0, ..., \tilde{J}_{k_q}, 0, ..., 0) \tag{3.40}$$

, und um die Notation beizubehalten ist

$$\tilde{J}_{k_i} \underset{\Delta\tilde{x}\to 0}{=} \frac{\delta e_k(\widehat{x} \boxplus \Delta\tilde{x})}{\delta\Delta\tilde{x}_{k_i}} \tag{3.41}$$

und das zu lösende Gleichungssystem wird zu

$$\tilde{H}\Delta\tilde{x}^* = -\tilde{b} \tag{3.42}$$

Die updates $\Delta\tilde{x}^*$ befinden sich in einer euklid'schen Umgebung um $\widehat{x}$ herum und müssen durch den $\boxplus$ Operator übersetzt werden.

$$x^* = \widehat{x} \boxplus \tilde{x}^* \tag{3.43}$$

3.3.2 Erhöhen der Robustheit

In Gleichung (3.12) fällt auf, dass $e_k(x)$ quadratischen Einfluss auf den Gesamtfehler hat, was die Fehlerfunktion anfällig gegenüber Ausreissern macht. Um dem entgegenzusteuern gibt es in G^2O die Möglichkeit, die F_k für $\sqrt{|F_k|} < b$ unverändert zu lassen, aber für $\sqrt{|F_k|} \geq b$ auf $2b\sqrt{|F_k|} - b^2$(Huber-Kostenfunktion) zu setzen, was in quadratischen Einfluss für kleine Fehler,

und linearem Einfluss für große Fehler resultiert. Tatsächlich wurde Gleichung (3.12) nicht modifiziert, sondern zunächst F_k wie gehabt berechnet, und dann durch eine gewichtete Version ersetzt.

$$F_k = (w_k e_k(x))^T \Sigma_k (w_k e_k(x)) \text{ mit} \tag{3.44}$$

$$w_k = \begin{cases} 1 & \text{für} \quad \sqrt{|F_k|} < b \\ \frac{\sqrt{2b\sqrt{|F_k|} - b^2}}{\sqrt{|F_k|}} & \text{für} \quad \sqrt{|F_k|} \geq b \end{cases} \tag{3.45}$$

Kapitel 4

Sensorik

4.1 RGBD-Kameras

Der Markt der RGBD-Kameras wird zurzeit von zwei Modellen dominiert. Eines wurde von Microsoft entwickelt, ist in Kombination mit der Spielekonsole XBox 360° des selben Herstellers benutzbar und trägt den Namen "Kinect". Das andere Modell ist von ASUS und hat den Namen "X-tion Pro Live". Da die technischen Unterschiede beider Modelle gering sind und in dieser Arbeit mit der ASUS X-tion Pro Live gearbeitet wurde, beziehen sich Aussagen über RGBD-Kameras im Folgenden, sofern nicht zusätzlich anders erwähnt, nur auf dieses Modell.

Abbildung 4.1: ASUS X-Tion Pro Live

Technische Daten(Herstellerangabe):
- Stromverbrauch: $<$ 2,5W
- Sinnvolle Entfernung zu Objekten während des Einsatzes: 0,8 - 3.5m. Ausserhalb dieses Bereiches wird die Tiefeninformation fehlerhaft.

- Sichtfeld: 58° horizontal; 45° vertikal; 70° diagonal
- Auflösung des Tiefensensors: 320x240(QVGA) bei 60Hz oder 640x480(VGA) bei 30Hz Bildwiederholrate
- Maximale Auflösung des RGB-Bildes: 1280x1024(SXGA)

Die Kamera besitzt eine USB 2.0-Schnittstelle, worüber sowohl der Datenstrom als auch die Energieversorgung laufen. Sie liefert im Betrieb stets Bildpaare, bestehend aus einem RGB-Bild und einem Tiefenbild, wobei letzteres schlicht jedem Bildpunkt eine Entfernung in Blickrichtung zuweist. Die Entfernungsmessung funktioniert über einen eingebauten, aktiven Infrarotsensor, der also Infrarotstrahlung aussendet und empfängt. Aus diesem Grund ist das Gerät nicht für Außenaufnahmen gedacht - die Infrarotstrahlung der Sonne würde tagsüber den Tiefensensor derart beeinflussen, dass die Tiefeninformation nahezu unbrauchbar wird.

Ideal wäre es, wenn der Aufnahmebeginn von RGB-Bild und Tiefenbild zeitgleich passieren würde. Da zwei Zeitpunkte in der Praxis nie exakt zusammenfallen wird es einen zeitlichen Versatz zwischen beiden Ereignissen geben. Auch hierzu waren keine Angaben des Herstellers zu finden. Allerdings lässt sich eine Obergrenze für den Versatz leicht festlegen. Wenn man eine RGBD-Kamera mit x Hz; $x \in \mathbb{N}^+$ betreibt, so beträgt der zeitliche Abstand des Aufnahmebeginns zweier aufeinanderfolgender RGB-Bilder etwa $\frac{1}{x}$sek. Dazwischen muss ein Tiefenbild aufgenommen werden, das entweder dem ersten, oder dem zweiten RGB-Bild zugeordnet wird. Andernfalls betrüge die Aufnahmefrequenz der Tiefenbilder weniger als x Hz. Daraus kann gefolgert werden, dass sich als Obergrenze für den angesprochenen Versatz $\frac{1}{x}$sek. festlegen lässt.

4.2 Extrinsische Kalibrierung

Seien die vier Kameras benannt mit K_1 bis K_4. Da sie keine überlappenden Sichtbereiche haben, ist es notwendig, auf andere Weise als die der direkten gegenseitigen Registrierung möglichst gute Näherungen für die Transformationen zwischen ihnen zu erhalten. Es existierte die Idee eine fünfte Kamera K_5 so über dem Kameraträger aufzuhängen, dass sie einen Sichtbereich hat, der mit den Sichtbereichen zweier Kameras K_i und K_j(i $\neq$ j) überlappt. Dann könnte man K_5 auf beide Kameras registrieren und erhielte $T_{K_i}^{K_5}$ und $T_{K_j}^{K_5}$, sowie $T_{K_i}^{K_j} = {T_{K_j}^{K_5}}^{-1} \cdot T_{K_i}^{K_5}$. Dann könnte man die Position der Zusatzkamera

ändern, so dass Teile ihres Sichtbereiches mit K_i und $K_k(k \neq i \neq j)$ teilt und dasselbe dort wiederholen.

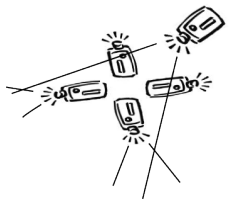


Abbildung 4.2: Kalibrierung mit einer Hilfskamera

Durch zwei solcher Positionsänderungen könnte man eine Kette bilden(z.B. von K_1 über K_2 und K_3 nach K_4) und könnte so die Transformation zwischen jedem Kamerapaar berechnen. Besser wäre hier allerdings auch gewesen, noch den Anfang und das Ende der Kette aufeinander zu registrieren, um die Effekte der Fehlerfortpflanzung zu minimieren. Diese Idee wurde aber nicht in die Tat umgesetzt, da der Anspruch bestand, auch für die Kalibrierung mit den vier gegebenen Kameras auszukommen, sollte hier aber der Vollständigkeit halber erwähnt werden.
Stattdessen wurden zunächst Kalibrieraufnahmen folgendermassen erzeugt: Der Kameraträger wurde auf einen stehenden Hohlkörper(Röhre) gesetzt, und in absoluter Ruhe von jeder Kamera ein Bild aufgenommen. Grundsätzlich wurde beim Aufnehmen erzwungen dass die Kameras in der Reihenfolge $(K_1, K_2, K_3, K_4, K_1, K_2, K_3, K_4...$ usw.) aufnehmen. Dann wurde der Hohlkörper um wenige Grad gedreht und anschliessend wieder in völliger Ruhelage vier Bilder aufgenommen. Nach diesem Schema wurde fortgefahren bis der Kameraträger eine 360° Drehung vollzogen hatte. Bei der Speicherung der Bilder muss darauf geachtet werden, die Information darüber mitzuführen, welches RGB-Bild zu welchem Tiefenbild gehört.
Dann wurden initial Transformationsmatrizen erstellt, die mittels Augenmaß grob geschätzt wurden. So befinden sich die Kameras näherungsweise in einer Ebene, also gibt es keine Translation in y-Richtung. Die Nickwinkel der Kameras wurden auf 30° vereinheitlicht, und die Gierwinkel wurden aufgrund der Anordnung auf dem Kameraträger auf 90° gesetzt. Translationen in x- und z-Richtung wurden mit einem Lineal abgemessen. Diese initialen Kalibriermatrizen wurden zusammen mit den Kalibrieraufnahmen dem SLAM-Verfahren übergeben, welches wegen der fehlenden Überlappung zunächst vier getrennte Kartenstücke erzeugt hat. Um einen Graph durch G^2O sinnvoll optimieren zu lassen, muss er zusammenhängend sein. Deswegen wurden die ersten vier Keyframes gegenseitig mit Kanten verbunden. Diese Kanten wurden äußerst "weich" gewählt, was bedeutet dass alle Varianzen auf den

Diagonalen der zugehörigen Kovarianzmatrizen sehr hoch($\simeq$ 0.2m, bzw. $\simeq$12°) gesetzt wurden. Im Laufe des Verfahrens trat eine Überschneidung der vier Kartenstücke auf. Das führte zu einer gegenseitigen Registrierung von Keyframes aus verschiedenen Kartenstücken und damit zum Einzug von "harten" Kanten zwischen den einzelnen Kartenteilen, was dazu führt, dass die weichen Kanten kaum noch Einfluss auf die Posen der Keyframes haben. An dieser Stelle sei erwähnt, dass eine derart grobe Initialisierung der Kalibriermatrizen bereits ausreichend ist, solange der fortgepflanzte Fehler zu den Zeitpunkten der erstmaligen gegenseitigen Registrierung zweier Kartenteile aufeinander noch nicht so stark angewachsen ist, dass die Registrierung fehlschlägt, oder überhaupt nicht durchgeführt wird. Durch die Graph-Optimierung(G^2O) korrigierten sich die Positionen der Keyframes und man konnte, nachdem genügend Messredundanz vorlag, deren Posen auslesen und aus den Differenzen die Transformationen der Kameras untereinander berechnen.

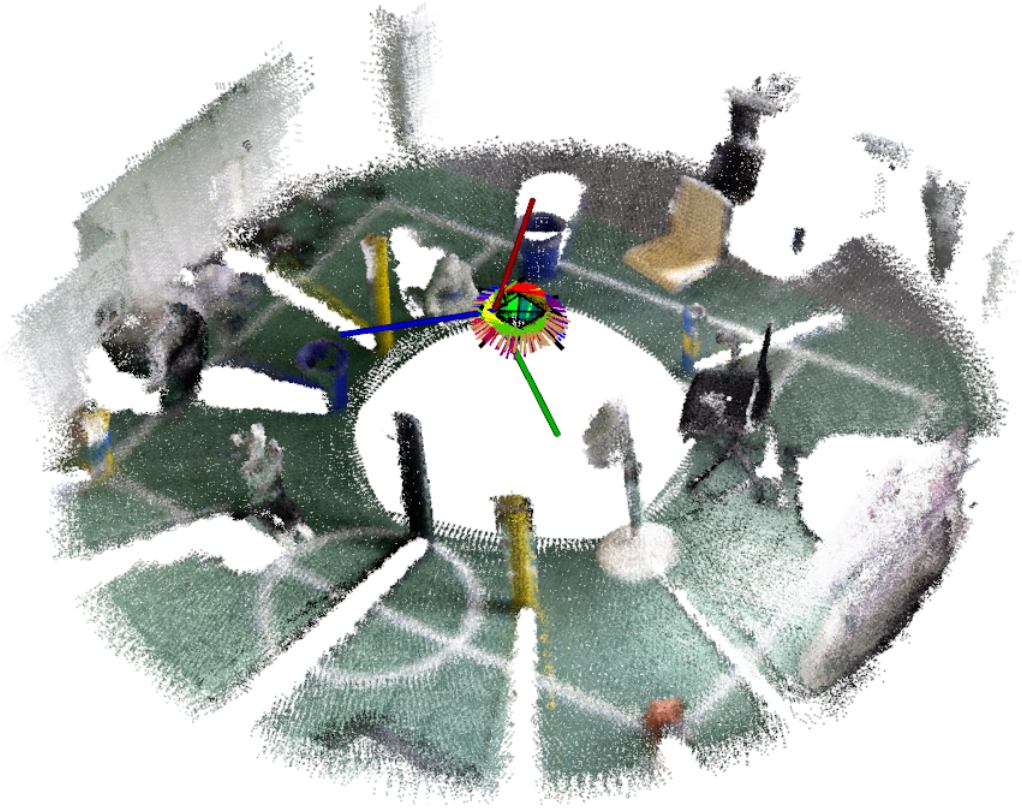


Abbildung 4.3: Kalibrierergebnis mit Blick in den freien Raum

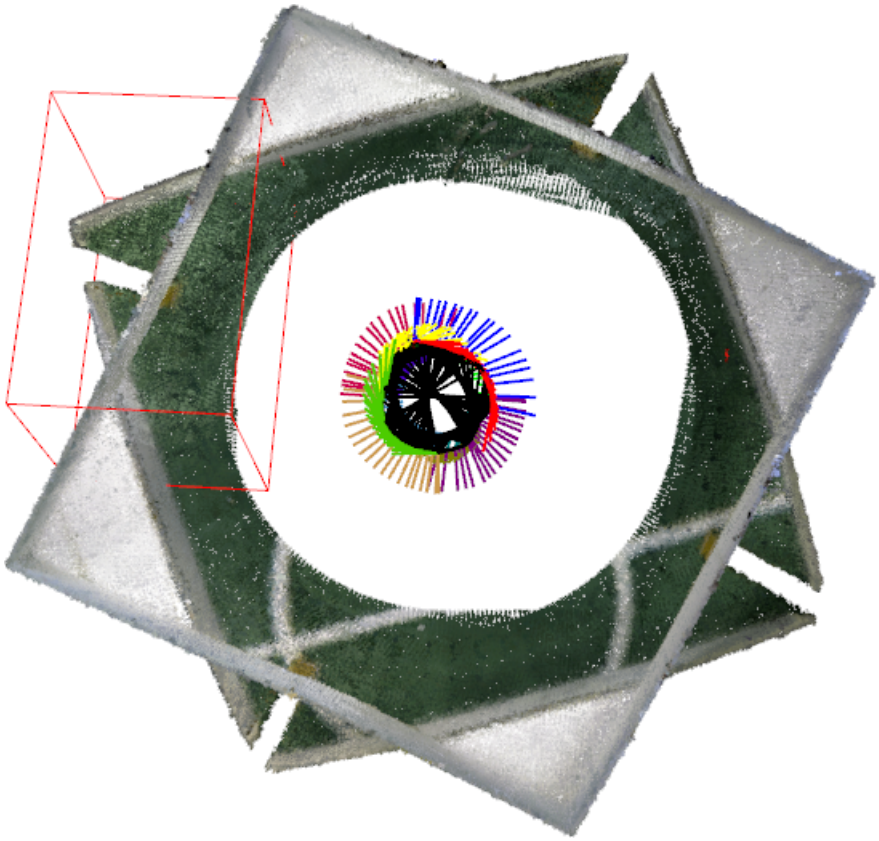


Abbildung 4.4: Kalibrierergebnis innerhalb einer um 45°versetzt übereinander gestellten Doppelraute aus Spanplatten

Kapitel 5

Das Kartierungsverfahren

5.1 Daten aufnehmen

Um ein möglichst breit gefächertes Sichtfeld zu erhalten, wurden vier ASUS X-tion Pro Live Kameras je im Winkel von ungefähr 90° zueinander versetzt auf einem tragbaren Metallträger angebracht. Da die Kameras laut Hersteller in ihrer zuverlässigen Reichweite auf 3,50m beschränkt sind, ein Mensch die Kameras aber in etwa 1,80m Höhe tragen würde, ist die Aufnahmerichtung nicht horizontal. Wäre sie es, und die Kamera blickt in einen großen Raum hinein, so würde das Tiefenbild unbrauchbar werden, da womöglich kein gesehenes Objekt näher als 3,50 m weit weg ist. Stattdessen sind die Kameras, jede für sich, um ca. 30° nach unten geneigt und mittels einer Epoxidharzverbindung fest fixiert. So soll auch die Wahrscheinlichkeit erhöht werden dass die Kameras nicht über strukturreiche Objekte wie z.B. Sitzmöbel, Türklinken, Pflanzen oder auch Wand-Boden-Kanten hinwegsehen, was die Robustheit der visuellen Odometrie erhöht. Diese Anordnung ist für den Rahmen dieser Arbeit sinnvoll, da die Träger des Gestells zur Zeit ausschließlich menschlich sind. Die Epoxidharzverbindungen können bei Bedarf jederzeit aufgebrochen, und die Positionen der Kameras verändert werden, z.B. für den Einsatz auf einem Roboter. Natürlich ist dann eine erneute Kalibrierung nötig.
Die Kameras sind zum einen durch Beschriftungen mit den Ziffern 1 - 4 versehen, zum anderen unterscheidet die Aufnahmesoftware zwischen den Kameraschnittstellen 1 - 4, und es muss vor jeder Aufnahme darauf geachtet werden, dass für i $\in \{1,2,3,4\}$ die Schnittstelle i mit der Kamera i verbunden ist. Das scheint notwendig zu sein, da wir bisher keine eindeutige, softwareseitig auslesbare Identifikationsnummer in den Geräten gefunden haben.

Abbildung 5.1: Rundumsicht durch vier Kameras

Die Schnittstelle zwischen PC und Kamera wurde über einen Treiber realisiert, der von der Organisation OpenNI [28] entwickelt wurde und den Namen "OpenNIGrabber" trägt. Das "open" in OpenNI ist eine gerne benutzte Bezeichnung für quelloffene Software. Das "NI" ist die Abkürzung für "natural interaction" und steht im weiten Sinne für Datenaustausch zwischen Natur und Maschine ohne Benutzung üblicher Peripheriegeräte wie Maus, Tastatur oder Gamepad, sondern durch Non-Touch-Geräte wie z.B. Kameras oder Mikrophone.
Um möglichst viel Umgebungsinformation zu erhalten, wurde die Aufnahmeauflösung auf VGA(640 x 480) eingestellt. Die Datenmenge, die bei dieser Einstellung und 30Hz Aufnahmefrequenz über den Bus geschickt wird ist so groß *insert genaue Angaben here*, dass der Betrieb zweier Kameras über denselben Bus bei Verwendung von USB 2.0 bereits mit Geschwindigkeitseinbußen einher geht. Gelöst wurde dieser Umstand durch Einsatz eines Laptops, der über vier USB 3.0 Ports verfügt.
Zur deutlichen Erleichterung der Entwicklungsarbeit, sowie zur Reproduzierbarkeit der Programmeingabe, wurde offline gearbeitet. Ein erfolgreicher Aufnahmeprozess schreibt einen Datensatz von RGB- und Tiefenbildern auf die Festplatte, sowie einer Datei, die Informationen enthält, welches Bildpaar von welcher Kamera stammt. Damit die Daten geordnet vorliegen, wurde sichergestellt, dass die Daten der Kameras in der Reihenfolge der

Kamera-IDs(1,2,3,4,1,2,3,4,usw.) weggeschrieben werden. Diese Bedingung erzwingt eine Verzögerung, da eine Aufnahme von z.B. Kamera 3 immer dann verworfen wird, wenn nicht vorher Kamera 2 an der Reihe war. Eine vielfach wiederholte Kontrolle der Frequenz, in der die Aufnahmen weggeschrieben werden, hat ergeben, dass diese in der Regel 9Hz pro Kamera nicht unterschreitet und im Durchschnitt etwa 14Hz beträgt.

5.2 Vorverarbeitung der Daten

Die Eingabe an das Programm besteht in der Hauptsache aus dem übergebenen Bilderdatensatz(siehe Kapitel 5.1 ("Daten aufnehmen")). Um die Multi Resolution Surfel Map-Datenstruktur nutzen zu können, ist noch etwas Vorarbeit nötig. Zuerst werden vier Bildpaare nacheinander eingelesen und Punktwolken daraus berechnet. Ein Raumpunkt P entsteht, indem ein Pixel aus dem RGB-Bild durch eine Gerade zwischen Brennpunkt und Pixelposition in den Raum projiziert wird. P befindet sich auf der Gerade an dem Punkt, der vor der Kamera liegt, und den passenden Abstand aus dem Tiefenbild hat. Sei (C_x, C_y) der Bildmittelpunkt, f $\simeq$ 1,9mm die Brennweite der Kamera, (x,y) die Koordinaten des Pixels im Bild, w sei die Breite und h die Höhe der Bilder. Der Bildmittelpunkt ist leicht ermittelt:

$$\begin{aligned} C_x &:= \frac{w}{2} \\ C_y &:= \frac{h}{2}. \end{aligned} \tag{5.1}$$

P.z wird mit dem Wert aus dem Tiefenbild an der Stelle (x,y) belegt. Die anderen beiden Koordinaten errechnen sich aus:

$$\begin{aligned} P.x &:= (x - C_x) \cdot P.z \cdot f \\ P.y &:= (y - C_y) \cdot P.z \cdot f. \end{aligned} \tag{5.2}$$

P erhält zusätzlich die Farbinformation des Pixels(x,y) aus dem RGB-Bild. Als Datentyp zur Speicherung der Punktwolken wurde "PointXYZRGB" aus der "Point Cloud Library"(PCL) [30] verwendet. Dann werden vier leere MRSMs erstellt und bekommen je eine Punktwolke übergeben, auf deren Basis Surfel erzeugt werden(siehe Kapitel 3.2.1 "Repräsentation durch Multiresolution Surfel Maps "). Die Punktwolken werden nun nur noch im Rahmen des Graph-Pruning benötigt. Hier besteht nun die Wahl, die Punktwolken abzuspeichern, um während des Prunings schneller darauf zugreifen zu können. Das gewählte Vorgehen war jedoch, sie zugunsten des Speicherplatzes zu verwerfen und im Falle des Prunings neu zu erzeugen.

5.3 Karte erzeugen und verwalten

Nach dem Vorverarbeitungsschritt liegen vier MRSMs vor, die Karteninformation in Form farbiger Surfel enthalten. Eine stichprobenartige Kontrolle der Aufnahmefrequenz lässt darauf schließen, dass das Zeitintervall, in dem alle vier Aufnahmen gemacht wurden, nicht größer als $\frac{1}{9}$sek. ist. Dies ist begründet durch die Beobachtung, dass während jeder Frequenzkontrolle die Aufnahmefrequenz aller Kameras höher als 9 Hz war. Die MRSMs sind nummeriert von 1 bis 4, und die MRSM mit Nummer $i \in \{1, 2, 3, 4\}$ enthält die Daten aus Kamera i. Ziel ist, einen SLAM-Graphen zu erstellen. Dies ist ein topologischer Graph, dessen Knoten besuchte Posen in sechs Freiheitsgraden, dargestellt durch eine Transformationsmatrix, repräsentieren, und dessen Kanten die Unsicherheiten der gegenseitigen Lage adjazenter Knoten in Form von 6x6 Kovarianzmatrizen tragen. Zusätzlich zum Graph wird eine Liste mitgeführt, die für jeden Knoten im Graph eine Instanz einer Klasse "Keyframe" abspeichert. Ein Keyframe enthält eine MRSM, sowie den Index des Knotens, zu dem sie gehört. Analog dazu ist in jedem Knoten im Graph der Index des ihm zugeordneten Keyframes abgespeichert. Der Knotenindex und der zugehörige Keyframeindex sind so lange identisch, bis das Graph-Pruning einsetzt. Danach gilt dies im Allgemeinen nicht mehr.
Zu Beginn ist der Graph leer. Dann werden vier Knoten im Graphen erzeugt. Der erste Knoten wird als Ursprung des globalen Koordinatensystems festgelegt, und dieser Knoten erhält den Vermerk "fixiert", was bedeutet, dass seine Position unveränderlich wird. Er dient sozusagen als Ankerknoten. Aus der Kamerakalibrierung(siehe Kapitel 4.2 ("Extrinsische Kalibrierung")) liegt eine Schätzung der relativen Lage der Kameras vor. Daraus ergeben sich direkt die Posen der anderen drei Knoten. Durch das Einfügen von sechs Kanten werden die vier Knoten zu einer Clique erweitert. Dabei wird dem Umstand Rechnung getragen, dass die Aufnahmezeitpunkte nicht zeitgleich passiert sein können, sondern im Allgemeinen ein zeitlicher Versatz besteht. Ohne diesen Versatz wäre es möglich, die "Härte" dieser künstlichen(= nicht aus einem Registrierungsprozess resultierenden) Kanten an die Genauigkeit der Kalibrierergebnisse anzupassen. Da sich jedoch der Kameraträger zwischen den Aufnahmezeitpunkten zweier Kameras geringfügig fortbewegt haben kann, sollte diese Einstellung deutlich "weicher" gewählt werden. Die Ausdrücke "hart" und "weich" stehen für kleinere, bzw. größere Varianzwerte auf der Hauptdiagonalen der Kovarianzmatrix. Sie versinnbildlichen die Analogie zwischen Kanten im Graphen und Spiralfedern, mit denen punktförmige Massen verbunden sind. Möchte man die Federn nicht überdehnen, so lässt sich eine solche Masse nur innerhalb des gemeinsamen Spielraumes aller an ihr befestigen Federn bewegen. Es sorgen

ausserdem härtere Federn für weniger Spielraum als weichere, analog zu den Kanten im Graphen.

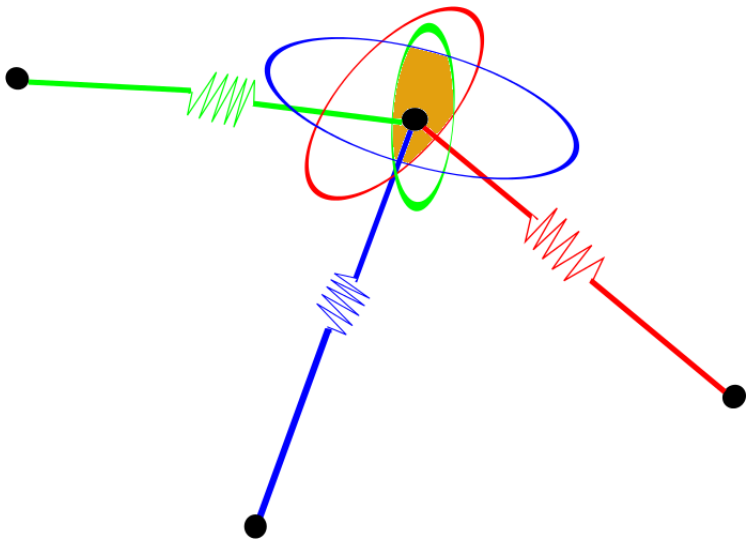


Abbildung 5.2: Beispielbild für den 2D-Fall: Die schwarzen Punkte stellen Kugeln dar. Die Äußeren seien der Übersichtlichkeit halber fixiert, während die innere Kugel beweglich ist. Zwischen den Kugeln sind Federn unterschiedlicher Härte gespannt. Will man eine Feder nicht überdehnen, so darf ihr bewegliches Ende nur innerhalb der Ellipse in ihrer Farbe ausgelenkt werden. Soll keine Feder überdehnt werden, so darf die Kugel nur innerhalb des Schnittes der drei Ellipsen bewegt werden.

Da das Posentracking auf Basis der Registrierung gegen Referenzkeyframes abläuft, werden für jede Kamera jeweils der Index ihres Referenzkeyframes sowie eine Hypothese über die Transformation zwischen Kamerapose und Pose der Referenz mitgepflegt. Die Indizes der Referenzkeyframes sind die Indizes der, auf Basis der jeweiligen Kamera gerade erzeugten, Keyframes(an dieser Stelle: 0,1,2 und 3). Die Transformationen werden mit der Nulltransformation, dargestellt durch Einheitsmatrizen, belegt.
Dieser erste Schritt endet hier. Also werden die nächsten vier Bildpaare eingelesen und MRSMs daraus erzeugt. Nun ist der vorhandene Graph nicht leer, weshalb ein Posenversatz geschätzt werden kann. Das Schätzverfahren ist in der Lage, Kartenteile beliebiger und unterschiedlicher Größe aufeinander zu registrieren. Dazu übergibt man allgemein eine oder mehrere MRSMs, deren Daten den Startpunkt(Quelle) der Transformation repräsentieren, und ebenso eine oder mehrere MRSMs als Ziel der Transformation. Konkret werden hier die vier Referenzkeyframes als Quelle, sowie die aktuell erzeugten

MRSMs als Ziel übergeben. Als Ergebnis erhält man bei erfolgreicher Registrierung vier Transformationen. Diese sind die neuen Hypothesen für die Positionen der Kameras relativ zu ihren Referenzkeyframes. Zudem werden Funktionen aufgerufen, die zu jeder dieser Transformationen eine Kovarianzmatrix und einen Wert für die Matchingloglikelihood schätzt und zurückgibt. Üblicherweise funktioniert eine Transformationsschätzung umso besser, je ähnlicher sich Quelle und Ziel sind, also je kleiner die zu schätzende Transformation ist, was in der Regel einen größeren Überlappungsbereich bedeutet. Daher ist es nötig, eine Methode einzuführen, die entscheidet, ob eine Transformation "klein" genug ist. Die Methode "poseIsClose" bekommt als Eingabe eine Transformationsmatrix T. Daraus extrahiert sie die Länge der Translation als Wurzel der Quadratsumme der x-, y- und z-Verschiebung, und den Drehwinkel aus dem Quaternion welches die Rotation aus T beschreibt. Sind Länge sowie Drehwinkel beide unterhalb voreingestellter Werte, so gibt die Funktion "true" wieder, andernfalls ist die Rückgabe "false". Analog dazu ist auch eine Funktion "poseIsFar" definiert, die auf die gleiche Weise die Länge und den Drehwinkel extrahiert, aber deutlich höhere Schwellwerte hat als poseIsClose, und prüft ob beide Werte von der übergebenen Transformation nicht unterschritten werden. Beispiele für typische Schwellwerte für poseIsClose sind 30cm und 20°, und für poseIsFar 70cm und 40°.
Als Indizien dafür, ob die Schätzung erfolgreich war wurden mehrere Kriterien definiert. So darf die Matchingloglikelihood einen Schwellwert nicht unterschreiten und die Varianzen in jedem Freiheitsgrad dürfen einen weiteren Schwellwert nicht überschreiten. Ausserdem wird davon ausgegangen, dass der Kameraträger sich in einem gewissen Geschwindigkeitsrahmen bewegt, d.h. gibt poseIsFar für die geschätzte Transformation true zurück oder gibt einer der anderen Tests ein unerwünschtes Ergebnis, so wird sie als fehlgeschlagen bewertet. In diesem Fall wird dieser Schritt beendet, keine Werte werden verändert, und es wird mit den nächsten vier Bildpaaren fortgefahren, in der Hoffnung, dass mit diesen ein besseres Resultat erzielt wird. Wird die Schätzung stattdessen als erfolgreich bewertet, so wird die Transformation an poseIsClose übergeben. Ein positives Ergebnis bedeutet die Annahme, dass der Kameraträger genügend Nähe zu Keyframes hat um weiterhin erfolgreich gegen diese registrieren zu können. In dem Fall werden die Hypothesen über die Transformationen zu den Referenzkeyframes mit den geschätzten Transformationen aktualisiert.
Geht die Prüfung auf poseIsClose negativ aus, so werden aus den MRSMs Keyframes erzeugt und als deren Repräsentanten dem Graphen je ein Knoten hinzugefügt. Sei r ein Referenzkeyframe zu einer Kamerapose c und U der Koordinatenursprung. Dann ergibt sich die Transformation vom entsprechen-

den Knoten zum Ursprung als Transformation der erzeugenden Kamera und diese durch Aneinanderhängen der Transformationen:

$$T_c^U := T_r^U \cdot T_c^r. \tag{5.3}$$

Auf diese Weise berechnen sich die Posen der neu eingefügten Knoten. Wie auch schon bei der Initialisierung des Graphen werden wieder sechs weiche Kanten eingefügt, so dass dieses Keyframequadrupel eine Clique bildet. Ausserdem werden weitere, deutlich wichtigere Kanten zwischen den neuen Keyframes und den Referenzkeyframes eingezogen. Deren Kovarianz wurde bereits während der Registrierung geschätzt und wird nun diesen Kanten zugewiesen. Die neuen Keyframes werden Referenzkeyframes ihrer jeweilig erzeugenden Kamera. Da sich die Kameras aktuell an exakt deren Pose befinden, sind die Transformationen zwischen Kameras und Referenzen nun auf die Identität zu setzen.

An diesem Punkt ist der Graph im wesentlichen aktualisiert. Was in diesem Schritt noch folgt sind die Graphoptimierung, eine Optimierung bezüglich der Referenzkeyframes, eine Verfeinerung der Kanteninformationen und die Suche nach Knotenpaaren, zwischen denen noch Kanten eingezogen werden können.

Die Graphoptimierung minimiert den Fehler im Graphen durch Veränderungen der Posen in den Knoten(siehe Kapitel 3.3 ("G^2O")). Visualisiert man die Karte, so macht sich dies deutlich bemerkbar. Z.B. passiert es häufig, dass ein Gegenstand vorher deutliche Doppelkanten hatte, die optisch vergleichbar mit Unschärfe in RGB-Bildern sind, weil zwei Keyframes relativ zueinander nicht korrekt lagen. Nach erfolgreicher Optimierung ist dieser Gegenstand eine Einheit mit scharfen Kanten.

Um nicht unnötig Speicherplatz zu verbrauchen, müssen die Referenzkeyframes aktuell gehalten werden. Dies verdeutlicht folgendes Beispiel: Man stelle sich vor, der Kameraträger vollzieht mehrfach hintereinander eine komplette Drehung um seine Längsachse. Auch nach der ersten vollständigen Drehung würden weiterhin im Abstand von bestimmten Drehwinkeln neue Keyframes eingefügt, da sich die Referenzkeyframes relativ zur Drehrichtung gesehen immer hinter den Kameras befinden. Das würde vermieden, wenn das Referenzkeyframe immer das Keyframe mit der geringsten "Differenz" zur Kamera wäre. Um dies zu realisieren werden die Posendifferenzen zwischen jeder Kamera und jedem Keyframe berechnet. Bei n Keyframes sind das

O(4n) Berechnungen. Sei k ein Keyframe.

$$T_c^k := T_U^k \cdot T_c^U = T_k^{U^{-1}} \cdot T_c^U \tag{5.4}$$

Analog zu den Funktionen poseIsClose und poseIsFar werden aus diesen Differenztransformationen Längen und Winkel extrahiert und deren Minima gesucht. Genauer gesagt: Ein Keyframe wird als "am nächsten" zu der jeweiligen Kamera definiert, wenn beide Werte unter den aktuellen Minima liegen. Das Keyframe, welches am Ende diesen Status hat, wird das neue Referenzkeyframe k_{next}. Gleichung (5.4) entsprechend müssen auch die Transformationen von den Kameras zu ihren Referenzen angepasst werden.

Da sich die Posen von bereits eingefügten Knoten noch ändern können, und die Schätzung der Kovarianzen die Posendifferenz beider beteiligter Knoten als Eingabe benötigt, folgt, dass sich das Schätzergebnis zu zwei verschiedenen Zeitpunkten voneinander unterscheiden kann. Daher wird an dieser Stelle eine Funktion "refineEdges" aufgerufen. Hierfür wird eine Kantenliste global mitgepflegt, in welche in jeder Iteration alle Kanten, von denen sich mindestens ein adjazenter Knoten bewegt hat, eingefügt werden. Aus dieser Liste werden in refineEdges zufällig einige Kanten ausgewählt, erneut geschätzt und anschliessend aus der Liste entfernt.

Für die Graphoptimierung ist es wichtig, dass nicht nur Kanten aus dem Tracking zu den Referenzkeyframes eingezogen werden, sondern auch zusätzliche Kanten entstehen um z.B. kleine Schleifen zu schliessen, oder Querverbindungen in der Trajektorie zu ziehen. Dies führt neue Bedingungen in den Graphen ein und kann für Posenkorrekturen verantwortlich sein. Ebenso erhöht dies die Robustheit des Graphen gegenüber schlechter registrierten Kanten. Daher wird in einer bestimmten Umgebung um die aktuellen Kameraschätzungen zufällig ein Knotenpaar gezogen. PoseIsFar von deren Transformationsdifferenz muss negativ sein, die beiden Knoten müssen einen bestimmten Prozentsatz an Überlappung aufweisen, und es darf nicht bereits eine Kante zwischen beiden Knoten existieren. Treffen diese Kriterien zu, so findet an dieser Stelle ein Registriervorgang statt. Für beide Knoten gilt: Ist er noch nicht an einem Mergingprozess beteiligt gewesen, so besitzt er genau drei künstliche Kanten. Die Nachbarn am anderen Ende dieser Kanten bilden mit dem ausgewählten Knoten zusammen ein Quadrupel. Dieses Quadrupel wird komplett dem Registrierungsprozess übergeben. Damit gibt es drei Fälle. Zwei bereits zusammengesetzte Knoten werden gegeneinander registriert, ein zusammengesetzter Knoten wird gegen vier nicht zusammengesetzte Knoten registriert, oder vier nicht zusammengesetzte Knoten

werden gegen vier weitere nicht zusammengesetzte Knoten registriert. Aus der Registrierung werden wieder die Matchingloglikelihood und die Kovarianzmatrix des ausgewählten Knotenpaares geschätzt. Erfüllen beide Werte den Test auf ihre Schwellwerte und die geschätzte Transformation ist nicht poseIsFar, so wird eine Kante zwischen beiden Knoten eingezogen. Waren weitere Knotenpaare an der Registrierung beteiligt, so werden sie an Hand der gleichen Kriterien geprüft und im positiven Fall zwischen ihnen ebenfalls Kanten eingezogen. Wurde insgesamt keine neue Kante eingezogen, so wird ein neues Knotenpaar gezogen und der Test wiederholt. Dies geht so lange bis entweder eine Kante gezogen wurde, oder eine bestimmte Anzahl an zufälligen Knotenpaaren gezogen wurde.

5.4 Graph-Pruning

Das Graph-Pruning bezeichnet den Vorgang, aus dem SLAM-Graphen zur Laufzeit Knoten und/oder Kanten zu entfernen. Die Vorteile dabei sind im Wesentlichen die Ersparnis von Speicherplatz und Rechenzeit. Entfernte Knoten müssen nicht mehr im Speicher gehalten werden. Das wird auf jedem Rechensystem relevant, sobald das kartierte Gebiet eine gewisse Größe überschreitet. Zudem hängt die Laufzeit vieler Prozesse von der Anzahl der Knoten ab. Eine Kürzung des Graphen hat, konkret bezogen auf den in dieser Arbeit beschriebenen Ansatz, Einfluss auf die Graph-Optimierung, die Suche nach der Möglichkeit, neue Kanten einzufügen, die Verfeinerung der Kanten sowie auf das Graph-Pruning ansich.
Will man Teile aus dem Graphen entfernen, steht eine Auswahl derselben voran. Sinnvolle Kriterien hierfür sind sicherlich der Informationsverlust durch das Entfernen, oder die lokale Dichte möglicher Kandidaten. Die Speicherung von Knoten verbraucht wesentlich mehr Speicherplatz als die Speicherung von Kanten. Daher liegt der Fokus normalerweise auf dem Entfernen von Knoten. Kanten, die vom Entfernungsprozeß betroffen sind, können getrennt behandelt werden, wie z.B. durch ihre Löschung, dem Einzug von Ersatzkanten, oder anderweitige Umverteilung der Kanteninformation.
Das, in dieser Arbeit verwendete, Pruning nutzt die Eigenschaften der zugrundeliegenden Multi Resolution Surfel Map Eigenschaft aus um Knoten nicht vollständig zu löschen, und somit die enthaltene Karteninformation gänzlich zu verlieren. Stattdessen wird hier ein Verfahren beschrieben, welches zunächst eine Knotengruppe auswählt, und anschliessend zu einem Knoten zusammenfasst ohne dass Karteninformation verloren geht.

5.4.1 Knoten auswählen

Die Positionen der Surfels innerhalb eines Knotens sind unveränderlich. Daher ist es wichtig, dass Knoten, die zur Zusammenfassung ausgewählt werden, zueinander gut lokalisiert sind. Das bedeutet eine hohe Sicherheit, oder äquivalent dazu eine geringe Unsicherheit ihrer gegenseitigen Posen. Die Unsicherheit in der gegenseitigen Pose zweier Knoten wird ausgedrückt durch die Kovarianzmatrix einer verbindenden Kante. Auf der Hauptdiagonalen stehen die Varianzen der sechs Freiheitsgrade. Sind diese alle sehr klein, besagt dies, dass die gegenseitige Pose als sehr sicher bewertet wird. Die Kante ist dann "hart". Je härter eine Kante ist, desto weniger Spielraum lässt sie auch den beiden benachbarten Knoten, sich in ihrer gegenseitigen Pose zu bewegen. Bei genügend Härte einer solchen Kante und der Vorraussetzung einer korrekten Schätzung ihrer Kovarianz, ist der Fehler in der gegenseitigen Lokalisierung nicht mehr sichtbar. Visualisiert man die beiden, zu den Knoten gehörigen, Kartenstücke, so ist dann keine Inkonsistenz mehr zu erkennen. Nur unter diesem Umstand ist es prinzipiell erst sinnvoll, Knoten zusammenzufassen, da sonst die bestehenden Inkonsistenzen fixiert und somit nicht mehr korrigiert werden. Das folgende Algorithmenpaar beschreibt eine effiziente Methode "chooseVertices" um eine Teilmenge aller Knoten auszuwählen, in der jedes Knotenpaar durch einen Pfad verbunden ist, auf dem alle Kanten diese Eigenschaft erfüllen.

Data : SLAM-Graph G = (V = $\{v_i\}$, E = $\{e_i\}$); Schwellwert T;
Untergrenze U
Result : $V' \subseteq V$, mit folgenden Eigenschaften: ;
- Der Graph G' = (V', E'), mit E' = $\{e_i\}$: e_i verbindet zwei Knoten aus V', ist zusammenhängend ;
- die Varianzen jedes Freiheitsgrades jeder dieser Kanten sind alle kleiner als T.

```
V' = { } ;
for i = 1; i < |E|; i++ do
    if Alle Varianzen von e_i < T then
        v_j, v_k = Knoten inzident zu e_i ;
        V' = V' ⋃ {v_j, v_k} ;
        continueSearch(e_i, v_j);
        continueSearch(e_i, v_k);
        if |V'| ≥ U then
            break ;
        end
        else
            V' = { } ;
        end
    end
end
return V' ;
```

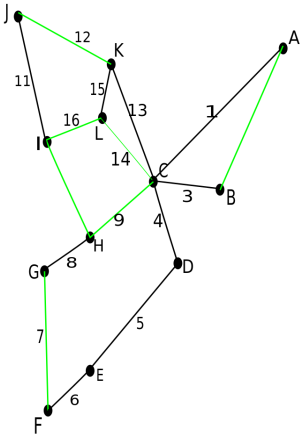
Algorithmus 1 : chooseVertices: Knoten zum Zusammenfassen auswählen

```
Data : SLAM-Graph G = (V = {v_i}, E = {e_i}); Schwellwert T;
       Obergrenze O; Kante e_i; Knoten v_j; Knotenmenge V'
if |V'| ≥ O then
    return;
end
for Alle Knoten e_l inzident zu v_j do
    if l ≠ i then
        v_k = Knoten an e_l am gegenüberliegenden Ende von v_j;
        if v_k ∉ V' then
            if Alle Varianzen von e_l < T then
                V' = V' ⋃ {v_k} ;
                continueSearch(e_l, v_k);
            end
        end
    end
end
return ;
```

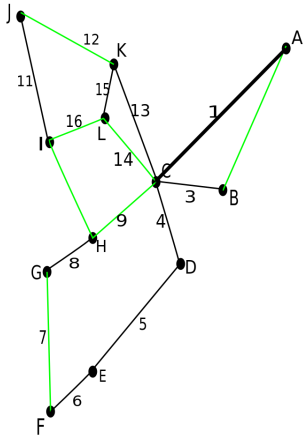
Algorithmus 2 : continueSearch: Funktion um das Auswählen von Knoten rekursiv fortzusetzen

Der Algorithmus iteriert durch die Kantenmenge so lange, bis entweder alle Kanten durchiteriert wurden, oder V' eine bestimmte Mindestgröße aufweist. Er prüft nacheinander für jede Kante, ob jeder der sechs Freiheitsgrade sicher genug ist, also die entsprechende Varianz klein genug ist. Trifft das zu, so ist diese Kante Ausgangspunkt einer Suche. Es wird dann mit den beiden Knoten, die sie verbindet, nacheinander continueSearch aufgerufen. Diese Funktion läuft rekursiv durch alle Kanten, die adjazent zu der übergebenen Kante sind, und prüft dort ebenfalls, ob die Varianzen klein genug sind. Trifft das zu, so wird der, durch diese Kante neu erreichte, Knoten zu V' hinzugefügt, sofern er nicht bereits enthalten ist. Nach dem Einfügen wird wieder continueSearch aufgerufen, und erhält diesmal den gerade eingefügten Knoten und die Kante, über die er erreicht wurde, um von dort aus die Suche fortzusetzen. Sollte V' bereits genügend Knoten enthalten($|V'| \geq O$), so wird die Funktion sofort nach ihrem Aufruf wieder verlassen. Sind jedoch Varianzen von der untersuchten Kante zu hoch, so verlässt continueSearch den aktuellen Aufruf, springt eine Rekursionstiefe

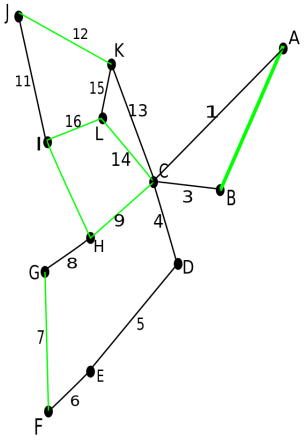
höher und setzt die Suche nach neuen Kanten ausgehend vom zuletzt besuchten Knoten fort.
Sind alle Funktionsaufrufe von continueSearch abgearbeitet, und V' besitzt nicht mindestens eine gewünschte Größe($|V'| < U$), so wird V' geleert, und die Suche nach Kanten, die Ausgangspunkt für eine neue Suche bilden, fortgesetzt. Ist aber $U \leq |V'| \leq O$, so wird V' unverändert zurückgegeben.
Durch den Umstand, dass bei der Suche nach neuen, zusammenzufassenden Knoten ausschliesslich Knoten in Betracht kommen, die in Adjazenz mit einem bereits in V' vorhandenen Knoten stehen, ist garantiert dass G' zusammenhängend ist.
Folgendes Beispiel soll den Algorithmus veranschaulichen. Kanten, bei denen die Varianz in jedem Freiheitsgrad klein genug ist, werden hier schwarz dargestellt, wogegen die anderen grün sind. Aktuell relevante Kanten sind fett gedruckt. Die Kanten sind von 1 bis 16 durchnummeriert und die Knoten tragen Bezeichner A bis L.



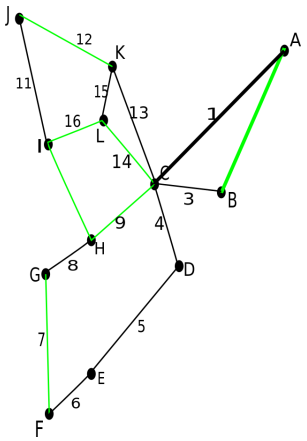
5.3: Der Graph vor der Suche.
V' = ∅



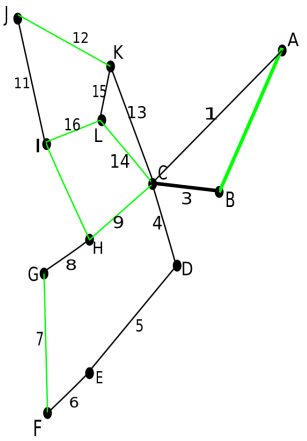
5.4: Prüfe Kante1, Ergebnis negativ.
V' = ∅



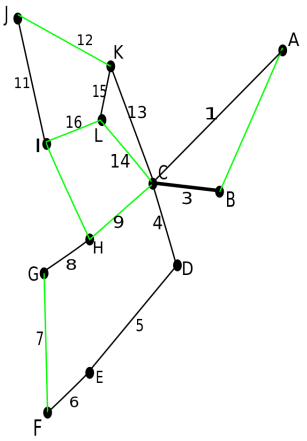
5.5: Prüfe Kante2, Ergebnis positiv.
V' = {A,B}



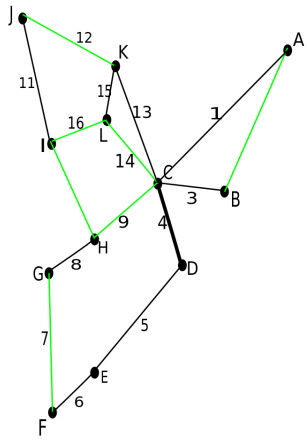
5.6: Prüfe Kante1, Ergebnis negativ.
V' = {A,B}



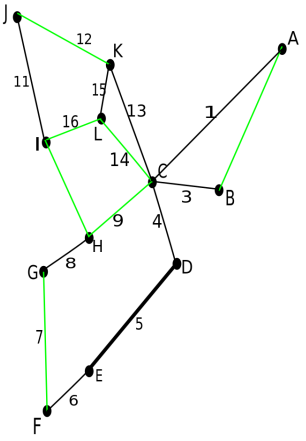
5.7: Prüfe Kante3, Ergebnis negativ.
V' = {A,B}.
Rekursionsende: $|V'| < U$, also leere V' und fahre fort.



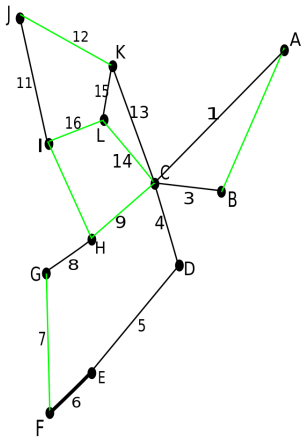
5.8: Prüfe Kante3, Ergebnis negativ.
V' = ∅.



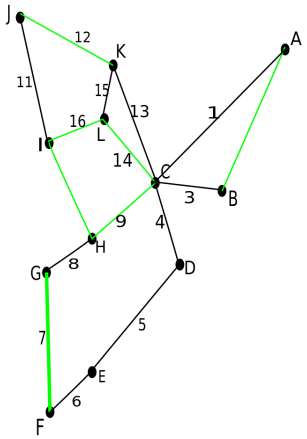
5.9: Prüfe Kante4, Ergebnis negativ.
V' = ∅.



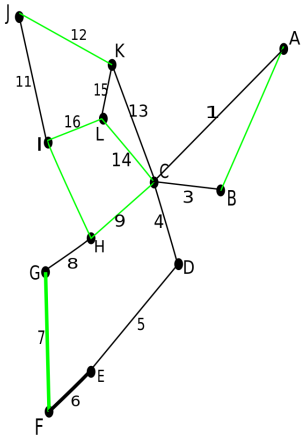
5.10: Prüfe Kante5, Ergebnis negativ.
V' = ∅.



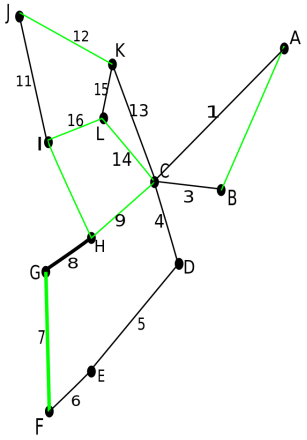
5.11: Prüfe Kante8, Ergebnis negativ.
V' = ∅.



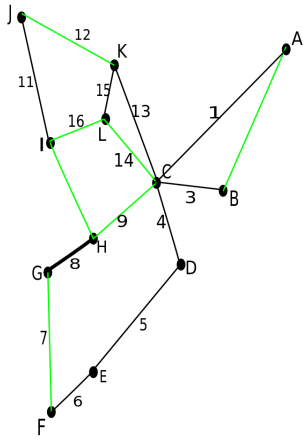
5.12: Prüfe Kante7, Ergebnis positiv.
V' = {F,G}



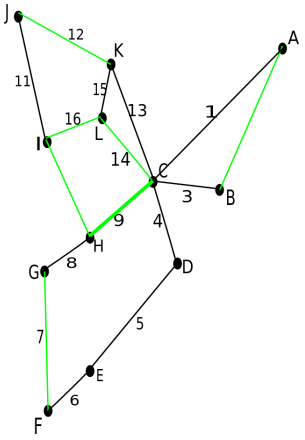
5.13: Prüfe Kante6, Ergebnis negativ.
V' = {F,G}



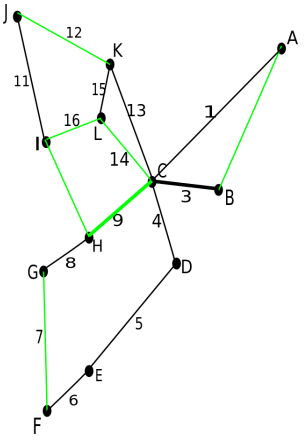
5.14: Prüfe Kante6, Ergebnis negativ.
V' = {F,G}.
Rekursionsende: $|V'| < U$, also leere V' und fahre fort.



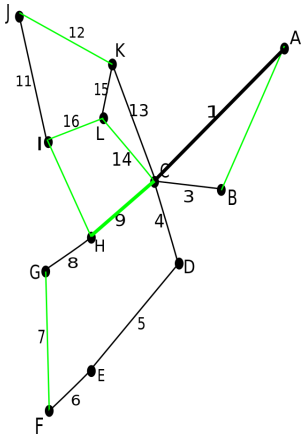
5.15: Prüfe Kante8, Ergebnis negativ.
V' = ∅



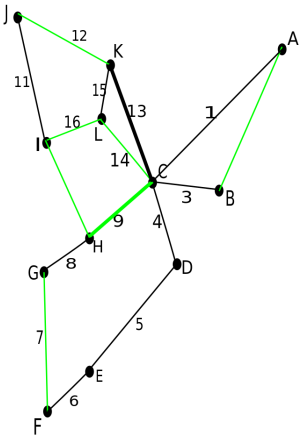
5.16: Prüfe Kante9, Ergebnis positiv.
V' = {C,H}



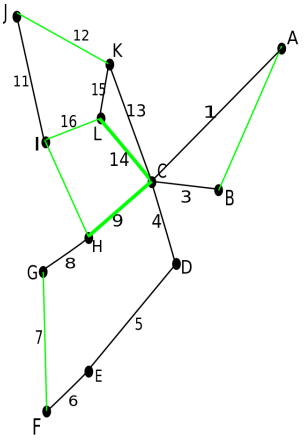
5.17: Prüfe Kante3, Ergebnis negativ.
V' = {C,H}



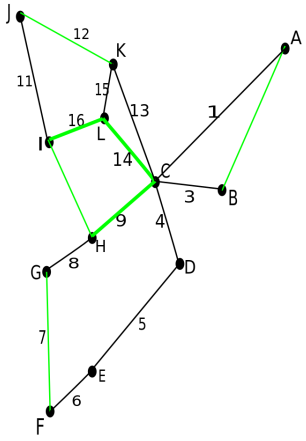
5.18: Prüfe Kante1, Ergebnis negativ.
V' = {C,H}



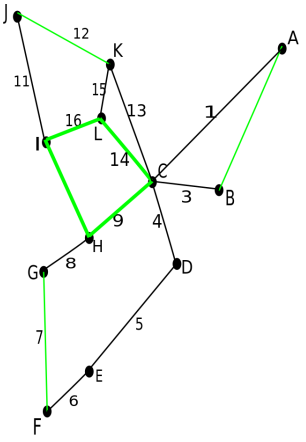
5.19: Prüfe Kante13, Ergebnis negativ.
V' = {C,H}



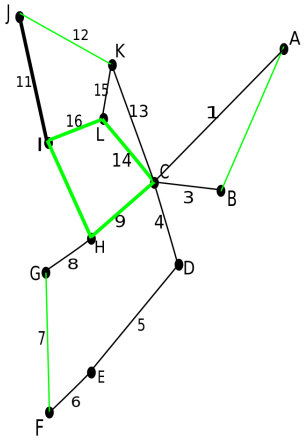
5.20: Prüfe Kante14, Ergebnis positiv.
V' = {C,H,L}



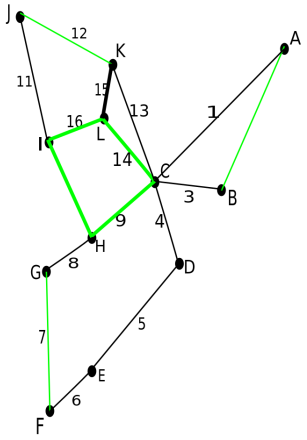
5.21: Prüfe Kante16, Ergebnis positiv.
V' = {C,H,L,I}



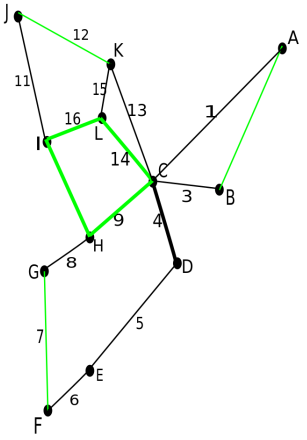
5.22: Prüfe Kante10, Ergebnis positiv.
V' = {C,H,L,I}



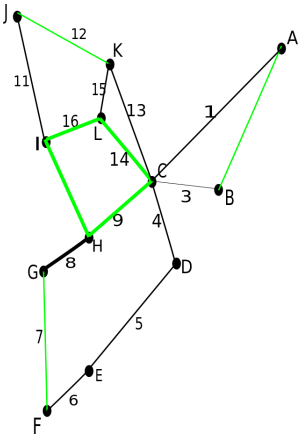
5.23: Prüfe Kante11, Ergebnis negativ.
V' = {C,H,L,I}



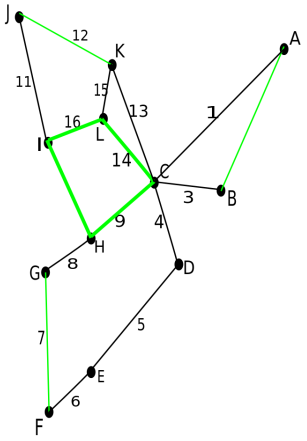
5.24: Prüfe Kante15, Ergebnis negativ.
V' = {C,H,L,I}



5.25: Prüfe Kante4, Ergebnis negativ.
V' = {C,H,L,I}



5.26: Prüfe Kante8, Ergebnis negativ.
V' = {C,H,L,I}



5.27: Rekursionsende: $|V'| \geq U$.
Gib V' = {C,H,L,I} zurück.

5.4.2 Knotengruppen zusammenfassen

War das Auswahlverfahren erfolgreich, so liegt eine Liste von Knoten vor, die zu einem Knoten zusammengefasst werden können. Um nicht zusammengehörige Keyframe-Vierer auseinander zu reissen, also Teile eines Vierers zusammenzufassen, und andere Teile nicht, wird die Bedingung gesetzt, dass ein zusammengesetztes Keyframe grundsätzlich aus ganzzahligen Vielfachen von Keyframe-Vierern bestehen soll. Um dies zu erreichen wird die Liste nun modifiziert. Zunächst wird in der Liste nach Knoten gesucht, die drei künstliche Kanten(als Kalibrierungsconstraints) besitzen. Bei Fund eines solchen Knotens, wird geprüft, ob die drei Knoten am Ende dieser Kanten ebenfalls in der Liste enthalten sind. Das bedeutet, dass ein komplettes Keyframequadrupel enthalten ist. Dann wird die Liste auf dieses Quadrupel beschränkt, und aus vier einzelnen Keyframes wird im Anschluss ein Einzelnes erstellt. Ist kein komplettes Keyframequadrupel enthalten, so wird nach Keyframes gesucht, die keine künstlichen Kanten haben. Dies können nur, im Rahmen des Graph-Prunings bereits zusammengefasste, Keyframes sein. Die Liste wird in diesem Fall um alle Keyframes bereinigt, die eine oder mehrere künstliche Kanten besitzen. Dann fasst das Pruning zwei oder mehr Keyframes zusammen, die zuvor bereits zusammengesetzt wurden, und damit nur komplette Keyframe-Vierer enthalten können. Sei $K := \{K_i\}; 1 \leq i \leq |K|$ die Liste der dazugehörigen Keyframes. Dazu wird ein neuer Knoten inklusive dem dazu gehörigen Keyframe K_{neu} erstellt. Der Knoten erhält die Pose des ersten Knotens in der Liste. Diese Wahl hat keinen bestimmten Grund - man hätte sich ebenso auf jeden anderen Knoten beziehen können. In jedem Keyframe des Graphen ist gespeichert, aus welchen Bildpaaren seine Daten erstellt wurden, da die Punktwolken aktuell nicht in den Keyframes gespeichert werden(siehe hierzu auch Kapitel 5.2 (“Vorverarbeitung der Daten”)). Ausserdem befindet sich dort zu jedem Bildpaar eine Transformationsmatrix. Vor dem ersten Pruningvorgang handelt es sich dabei um ein einziges Bildpaar, dem die Einheitsmatrix zugewiesen ist. Diese Transformationen beschreiben die Bewegungen vom lokalen Ursprung der jeweiligen MRSM zu der Pose der eingefügten Punktwolke. Diese Informationen sind nötig, um für K_{neu} die MRSM aus den passenden Punktwolken zu erstellen, sowie die Punkte innerhalb der MRSM an die richtige Stelle zu setzen. Aus allen, in den K_i vermerkten, Bildpaaren werden nun wieder Punktwolken erzeugt(siehe Kapitel 5.2 (“Vorverarbeitung der Daten”)). Diese Punktwolken sollen in die MRSM von K_{neu} eingefügt werden. Um dies korrekt zu vollziehen, ist es nötig, zuvor die passende Transformation bezüglich des Ursprungs von K_{neu} zu berechnen. Sei $P_{i,j}$ die j-te Punktwolke in K_i und U der globale Koordinatenursprung. Wir müssen $T_{K_{neu}}^{P_{i,j}}$ berechnen, um die jeweilige

Punktwolke an der richtigen Stelle einzusetzen.

$$\begin{aligned} T_{K_{neu}}^{K_i} &= T_U^{K_i} \cdot T_{K_{neu}}^U = T_U^{K_i} \cdot T_U^{K_{neu}{}^{-1}} \\ T_{K_{neu}}^{P_{i,j}} &= T_{K_i}^{P_{i,j}} \cdot T_{K_{neu}}^{K_i} \end{aligned} \tag{5.5}$$

Das neue Keyframe ist nun fertig. Als Nächstes werden alle Kanten, die den ersten ausgewählten Knoten(dessen Pose nun K_{neu} übernommen hat) mit nicht ausgewählten Knoten verbinden, so umgebogen, dass sie mit K_{neu} anstelle des ersten ausgewählten Knotens verbunden sind. Alle Kanten, die mit weiteren ausgewählten Knoten verbunden sind, sowie alle ausgewählten Knoten und ihre jeweiligen Keyframes werden gelöscht. Um die Konsistenz zu erhalten muss vorher geprüft werden, ob unter den gelöschten Keyframes Referenzkeyframes sind. In diesem Fall müssen die zugehörigen Kamerapositionen rekonstruiert werden. Dann folgt unter den verbleibenden Keyframes die Suche nach neuen Referenzen als die Keyframes, die im Sinne räumlicher Distanz und Winkel am nächsten zu den Kamerapositionen stehen. Anschliessend werden neue Kanten zwischen K_{neu} und dem Rest des Graphen eingefügt, was beides in Kapitel 5.3("Karte erzeugen und verwalten") beschrieben ist.
K_{neu} abzuspeichern verbraucht weniger Speicherplatz als K abzuspeichern. Das liegt an der Eigenschaft der Multi Resolution Surfel maps. Wird ein Punkt zu einem Voxel hinzugefügt, und sein Deskriptor passt zu den dort bereits enthaltenen Punkten, so resultiert dies in einem gemeinsamen Surfel. Als dieser Punkt noch in einer separaten MRSM gespeichert war, wurde er dem gleichen Surfel zugeordnet. Dieses Surfel gab es allerdings einmal in jeder MRSM, die den selben Objektpunkt abbildete. Die Grenzen dieser Ersparnis können angegeben werden durch Betrachtung der Extremfälle.
Obergrenze:
x Keyframes sollen zu einem Keyframe K_{neu} zusammengefasst werden, und alle diese Keyframes bilden exakt dasselbe ab, haben also den gleichen Sichtwinkel und einen Überlappungsbereich von 100%. Deren MRSMs wurden im Idealfall(ohne Messrauschen) aus identischen Punktwolken gebildet, und enthalten daher auch die selben Surfel. Jedes dieser Surfel wurde vor der Zusammenfassung in jeder der x MRSMs einmal abgespeichert. Die MRSM von K_{neu} enthält nach der Zusammenfassung die gleichen Informationen wie jedes der x Keyframes. Die Ersparnis kann hier also für das Zusammenfassen von x Keyframes mit $\frac{x}{x-1} \cdot 100\%$ nach oben abgeschätzt werden.
Untergrenze:
Um in der gegenseitigen Pose zweier Keyframes genau zu sein, müssen sie ein gewisses Minimum an Überlappung aufweisen. Angenommen, es existiere ein Schätzverfahren, welches mit so geringer Überlappung auskommt, dass sie

vernachlässigbar ist. In diesem Fall betrachten wir x zusammenzufassende Keyframes als disjunkt bezüglich des dargestellten Sichtbereiches. Dann können zwei Punkte, die bei der Erzeugung verschiedener Keyframes beteiligt waren, nicht in das selbe Voxel hineinfallen. Es wird also in K_{neu} die selbe Anzahl Surfel erzeugt, wie in allen x Keyframes zusammen. Folglich liegt die Ersparnis in dem Fall bei 0.
Eine in der Praxis funktionierende Untergrenze für den Überlappungsbereich, die auch bei den Experimenten benutzt wurde, ist 20%.

Kapitel 6

Experimente

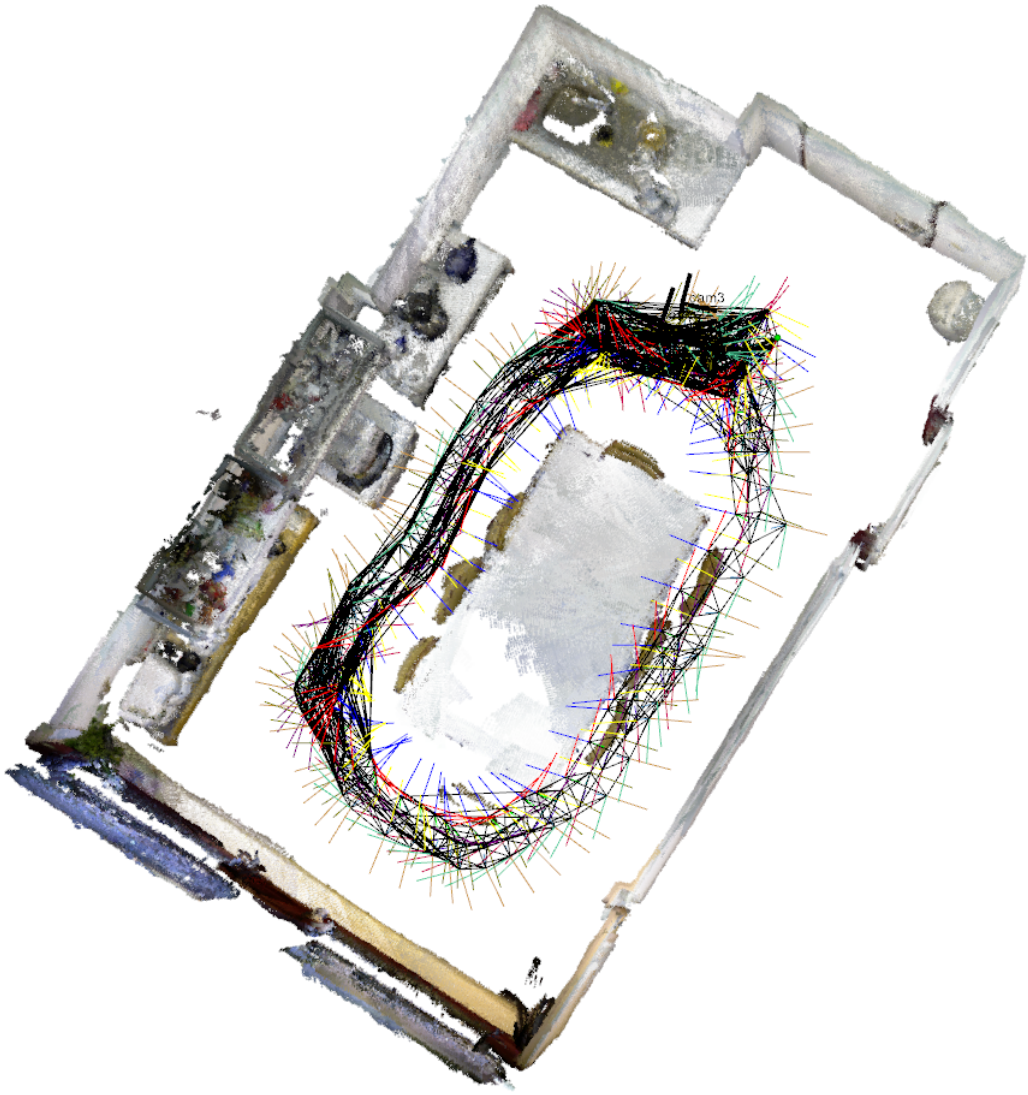
6.1 Kartierung

Während der Aufnahme von Bilddaten für die Kartierung wurde darauf geachtet sich langsam zu bewegen, damit möglichst wenig Bewegungsunschärfe entsteht. Ausserdem zeichnen sich die Wege, die bei der Aufnahme beschritten wurden, durch mehrfach vorhandene Kreise und Achten aus. So soll die Wahrscheinlichkeit erhöht werden dass Querverbindungen und kleine Schleifenschlüsse entstehen, die die Korrektheit der Karte verbessern. Getestet wurde das Kartierungsverfahren in Räumen der Abteilung VI des Instituts für Informatik der rheinischen Friedrich-Wilhelms-Universität Bonn.

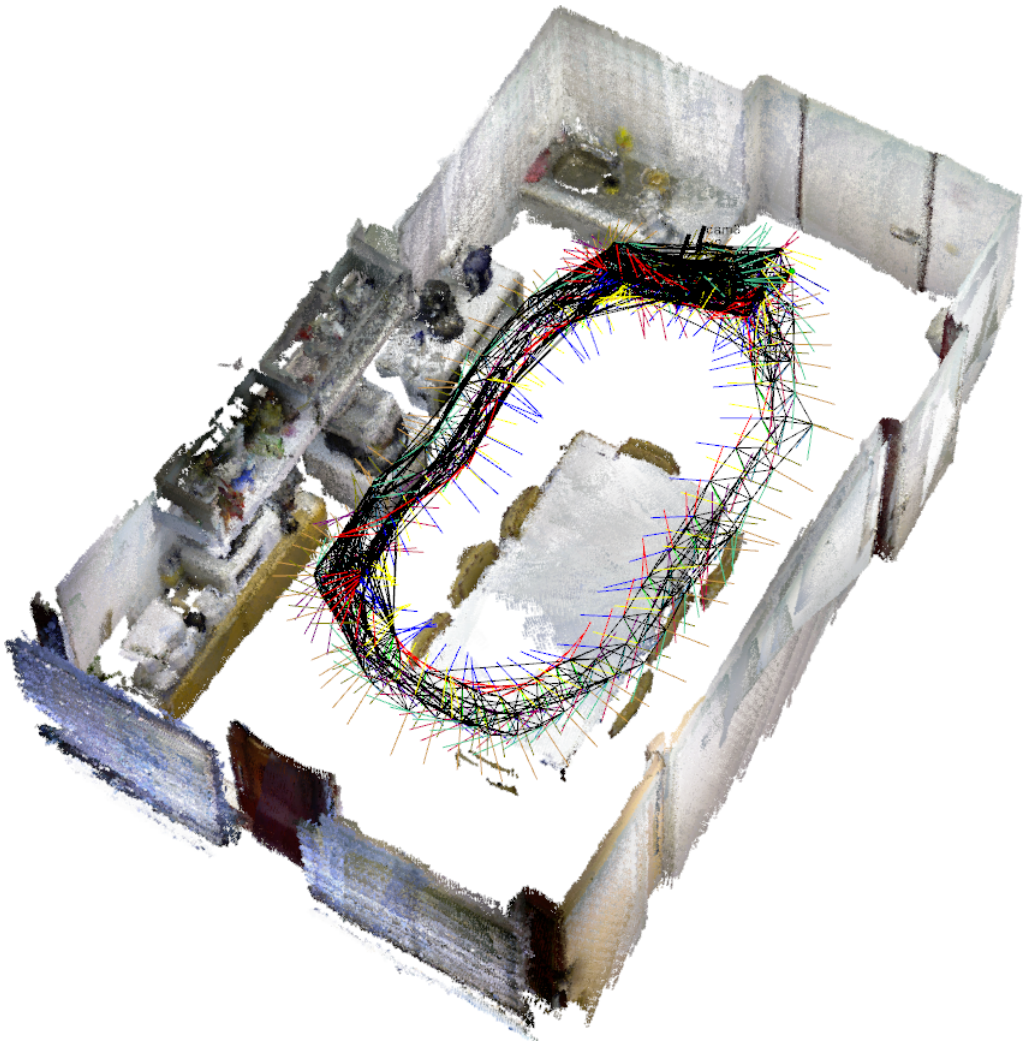
Die Abteilungsküche(s. Abb. 6.3 und 6.4) zählt zu den einfacheren Datensätzen. Sie ist verhältnismäßig klein, weist nur wenig texturarme Flächen auf und dort befinden sich ein Schrank und eine Spüle, die viele Details und wenig Symmetrie in die Karte mit einbringen. Um einen Eindruck dieser Räumlichkeit zu geben sind in Abbildung 6.1 und Abbildung 6.2 RGB-Bilder aus den Datensätzen dargestellt.

6.1: Ein Bild aus den Datensätzen stellt eine Hälfte der Abteilungsküche dar.

6.2: Ein Bild aus den Datensätzen stellt die andere Hälfte der Abteilungsküche dar.

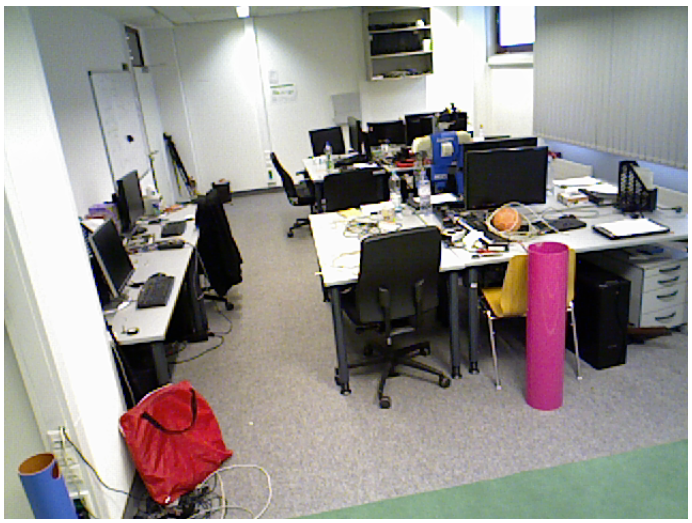


6.3: Die Abteilungsküche aus der Vogelperspektive

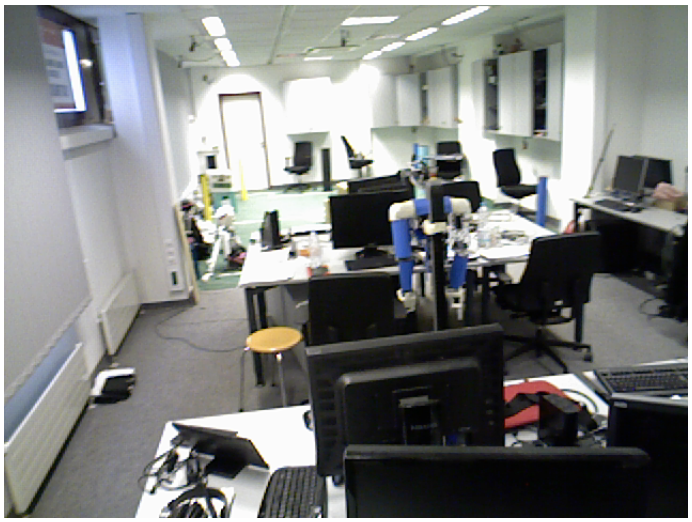


6.4: Die Abteilungsküche von schräg oben

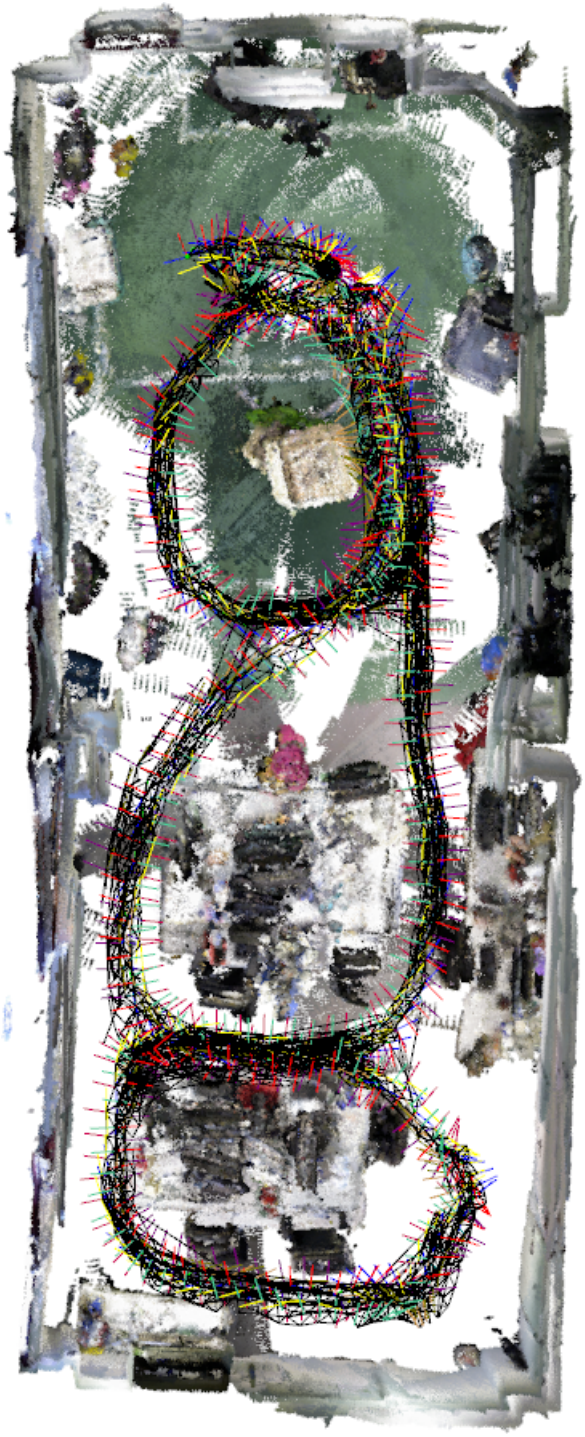
Es gibt in der Abteilung ein Robotiklabor(s. Abb. 6.7 und 6.8), welches auf der einen Seite aus einem Trainingsfußballfeld für Roboterfußball besteht und auf der anderen Seite aus einem Arrangement von Arbeitsplätzen, bestehend aus Tischen, Rechnern und Bildschirmen. Dieser Datensatz hat einen erhöhten Schwierigkeitsgrad, da die zurückgelegten Wege z.T. sehr nah an Jalousien vorbeiführen, die, ebenso wie einige Wandbereiche, größere texturarme Flächen darstellen und das Labor ein großer Raum von etwa 13m * 5m ist. Einen Eindruck des CI-Labors geben die Abbildungen 6.5 und 6.6 aus den Datensätzen.



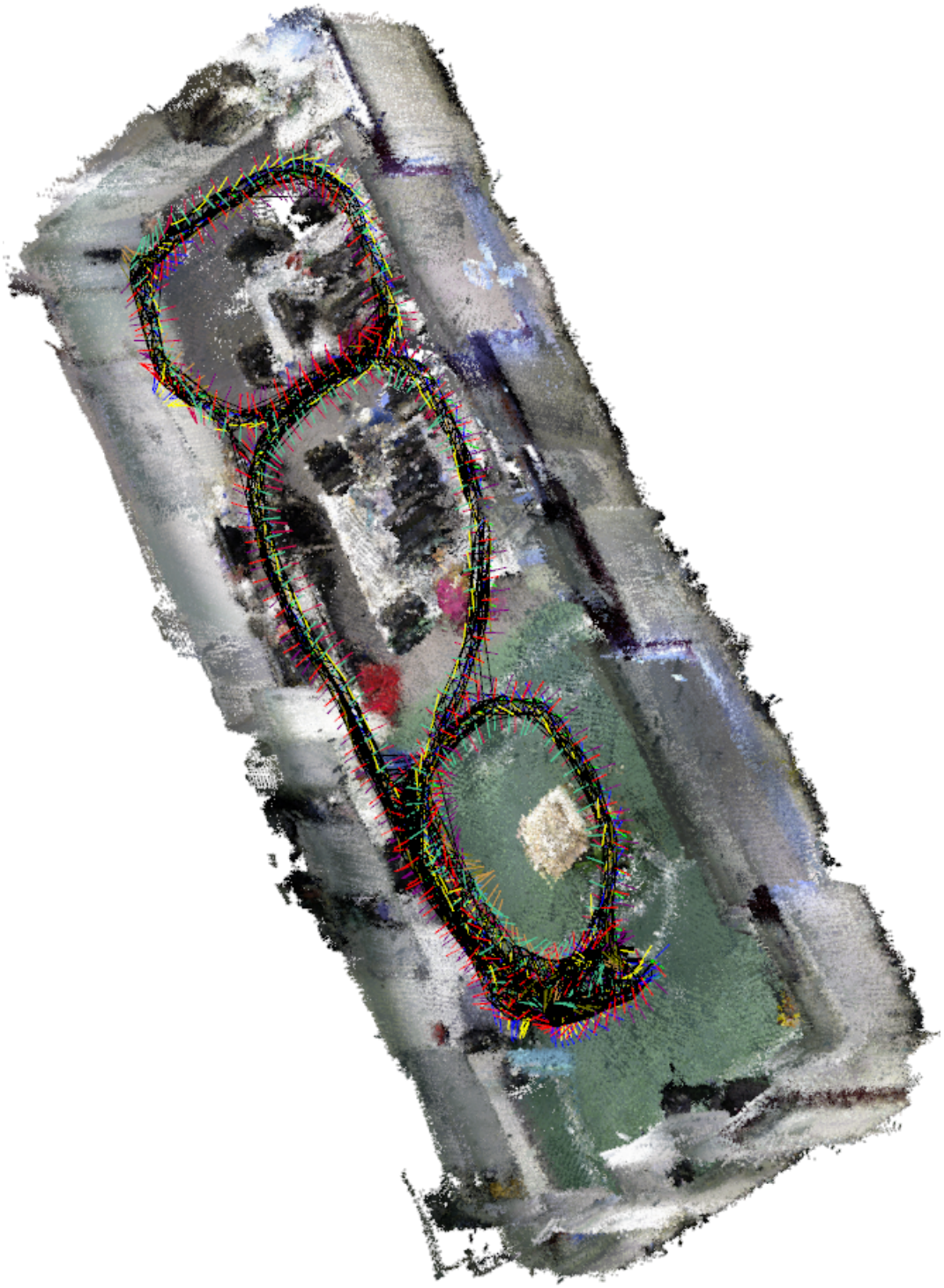
6.5: Ein Bild aus den Datensätzen stellt eine Hälfte des Robotiklabors dar.



6.6: Ein Bild aus den Datensätzen stellt die andere Hälfte des Robotiklabors dar.

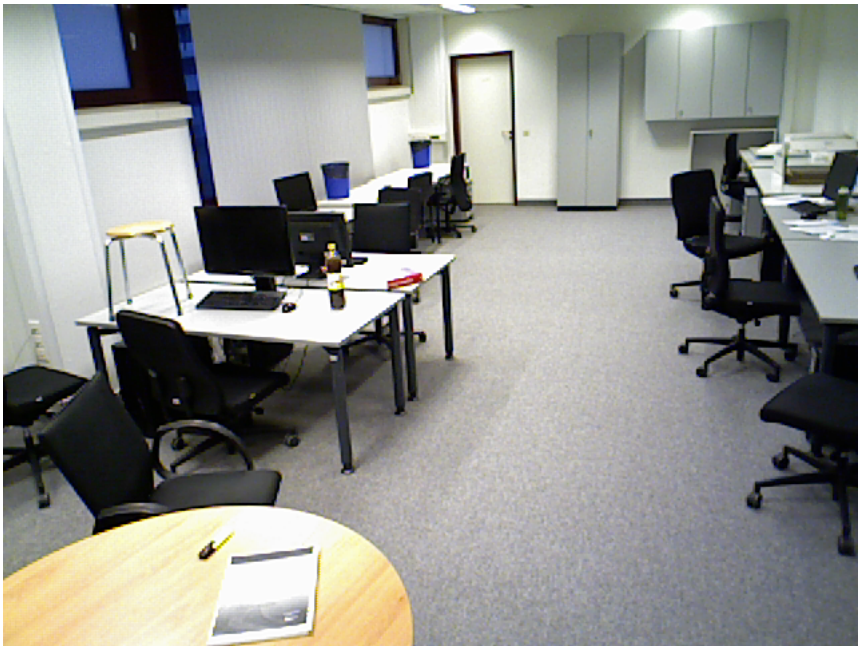


6.7: Das Robotiklabor aus der Vogelperspektive

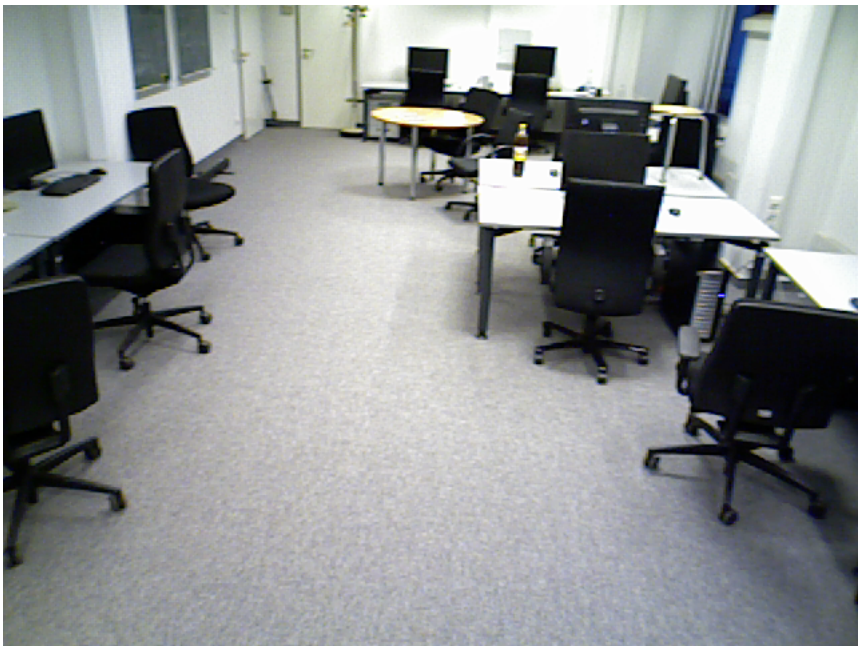


6.8: Das Robotiklabor von schräg oben

Von mittelmäßiger Schwierigkeit ist dagegen ein Datensatz aus dem CI-Labor(s. Abb. 6.11, 6.12, 6.13 und 6.14). Dies ist ein Raum von ähnlichen Ausmaßen wie das Robotiklabor, allerdings erlaubt es die Anordnung der Tische in diesem Raum, eine Trajektorie zu gehen, die nicht sehr nah an Wänden vorbei führt. Auch stehen dort weniger hohe Gegenstände im Raum, was die Häufigkeit von Tiefensprüngen an virtuellen Kanten reduziert. Auch hier sollen Abbildungen 6.9 und 6.10 aus den Datensätzen einen Eindruck verschaffen.



6.9: Ein Bild aus den Datensätzen stellt eine Hälfte des CI-Labors dar.



6.10: Ein Bild aus den Datensätzen stellt die andere Hälfte des CI-Labors dar.

Um Angaben zur Genauigkeit der Kartierung machen zu können, die nicht rein visueller Natur sind sondern Zahlen liefern, wurden Folgendes gemacht: Nach einem abgeschlossenen Kartierungsvorgang in einem dieser Räume wurden klar definierte Strecken bestimmt wie z.B. Tischbreiten oder Raumlängen. Die Positionen beider Enden jeder Strecke wurden aus der Karte abgefragt, und der euklid'sche Abstand daraus berechnet. Dies ist die vom Verfahren ermittelte Streckenlänge. Im Anschluss wurden die Strecken in der realen Welt mit einem Maßband vermessen und mit den Strecken aus dem Kartierungsverfahren verglichen.

Gemessene Strecke	**Länge(Karte)**	**Länge(Maßband)**
Länge des Küchentisches	1,961m	1,995m
Breite des Küchentisches	0,970m	0,995m
Länge der gesamten Küche	5,276m	5,260m
Breite der gesamten Küche	3,219m	3,250m
Breite der Hängeschränke in der Küche	2,003m	2,004m
Höhe des Küchenregales mit der Kaffeemaschine	1,030m	1,000m
Breite des Küchenregales mit der Kaffeemaschine	1,007m	1,000m
Länge des ersten Raumabschnittes(CI-Labor)	5,223m	5,250m
Länge des kompletten CI-Labors	12,343m	12,770m
Breite des Schrankes rechts hinten(CI-Labor)	1,545m	1,585m
Länge der Fußballfeldbegrenzung	5,983m	6,060m
Breite der Fußballfeldbegrenzung	4,010m	4,040m
Zwischen Tisch und Wandvorsprung(Robotiklabor)	2,857m	2,850m

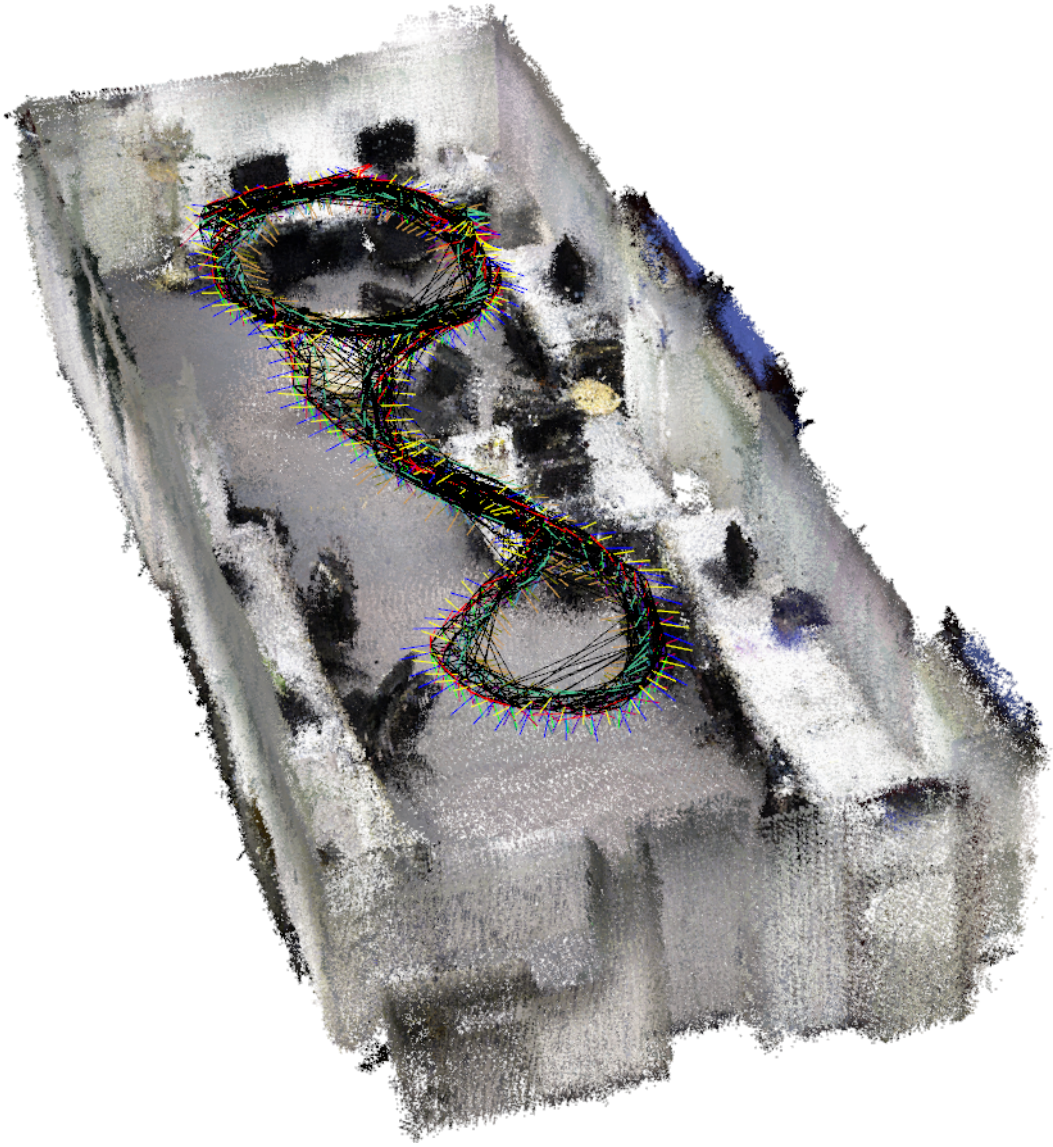
Tabelle 6.1: Vergleich zwischen Streckenlängen aus dem Kartierungsverfahren und real gemessenen Streckenlängen

6.2 Graph-Pruning

Das Graph-Pruning Verfahren ermittelt einen Teilgraph des SLAM-Graphen, in welchem die Varianzen jeder Kante unterhalb bestimmter Schwellwerte liegen(siehe Kapitel 5.4). Die mittleren Unsicherheiten der verschiedenen Freiheitsgrade unterscheiden sich deutlich. Daher wurde für jeden Freiheitsgrad ein eigener Schwellwert definiert. Die Knotenmenge des Teilgraphen wird daraufhin überprüft, ob sie einen komplettes Knotenquadrupel enthält, was bedeutet, daß keiner der vier Knoten bisher an einer Zusammenfassung beteiligt gewesen ist. Fällt der Test positiv aus, so wird die Liste der zusammenzufassenden Knoten auf diese vier reduziert. Andernfalls wird im Anschluss ausschliesslich nach Knoten gesucht, die bereits an einer Zusammenfassung beteiligt waren. Sind derer mindestens zwei in dem Teilgraphen enthalten, so wird die Liste der zusammenzufassenden Knoten auf diese reduziert.
Die Schwellwerte steuern das Pruningverhalten. Ist die Einstellung der Schwellwerte gering genug, so wird das Pruning nicht stattfinden, da ein für die Zusammenfassung geeigneter Teilgraph nicht gefunden werden kann.

6.11: Das CI-Labor aus der Vogelperspektive

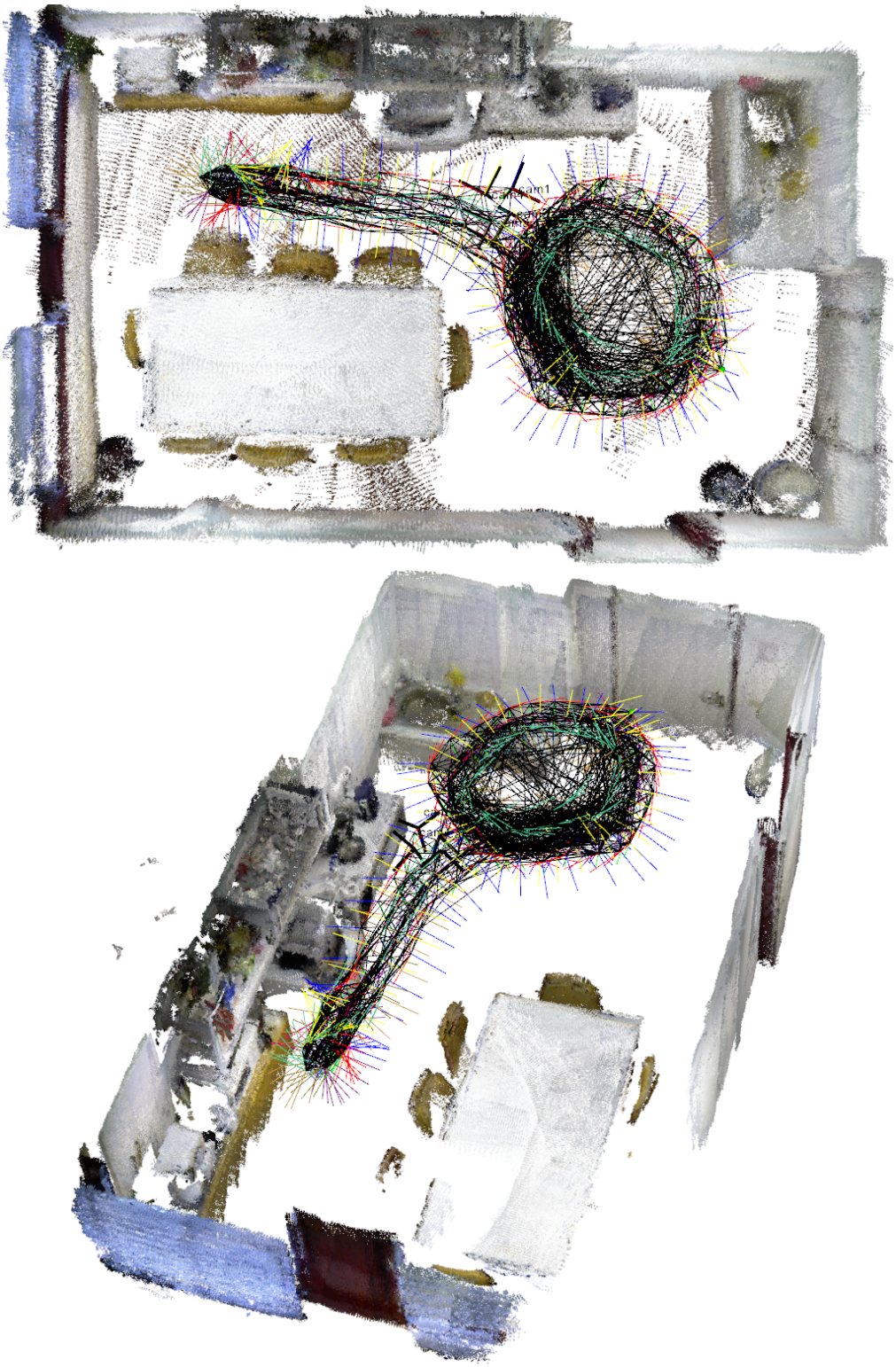
6.12: Das CI-Labor von schräg oben



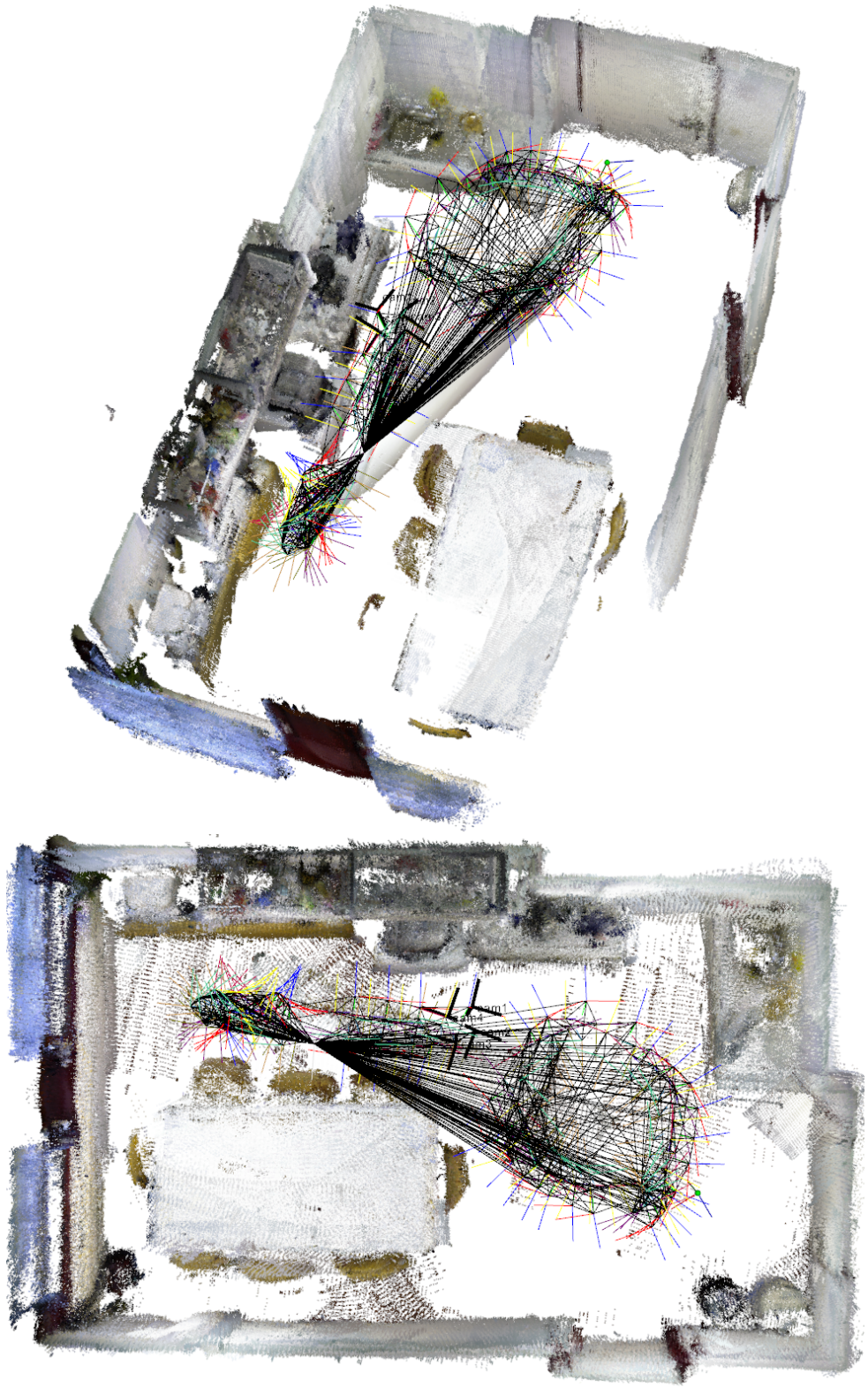
6.13: Eine weitere Ansicht des CI-Labors

6.14: Das CI-Labor aus der Vogelperspektive ohne Trajektorie

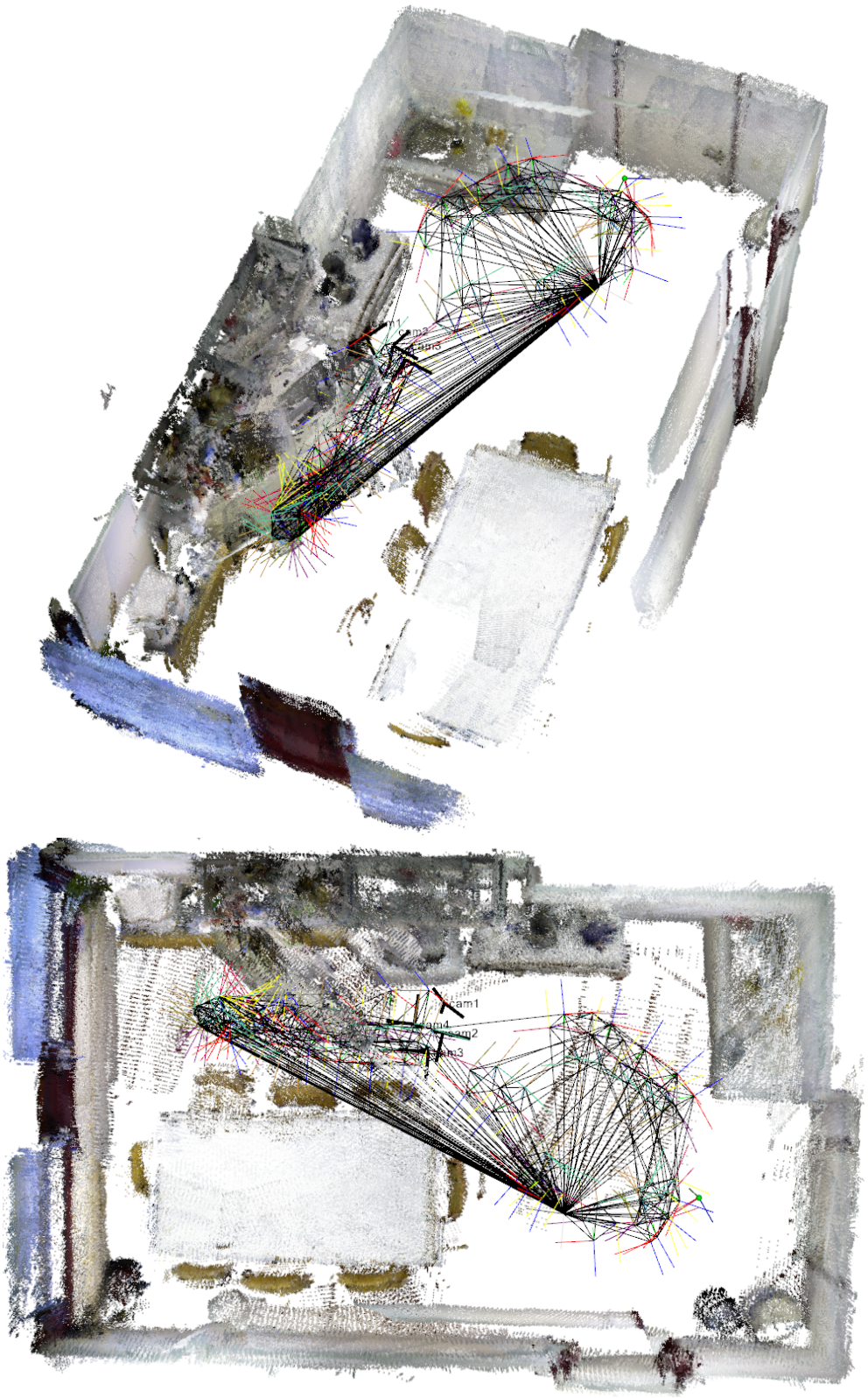
Ist sie dagegen hoch genug, so wird jede Kante des SLAM-Graphen die Bedingungen erfüllen und der fortlaufend wiederholte Pruningprozess tendiert dahin, den gesamten SLAM-Graphen in einem einzigen Knoten zusammenzufassen. Der erste Fall unterscheidet sich im Resultat nicht von einer Kartierung ohne Pruning. In letzterem Fall ist man sehr tolerant, was die Fehler in den Positionen der Keyframes vor dem Zusammenfassen angeht. So werden Posenfehler, die andernfalls noch hätten korrigiert werden können, fest in das zusammengefasste Keyframe eingeschrieben und somit von weiteren Korrekturen ausgeschlossen. Folglich muss an dieser Stelle ein Trade-off zwischen Kartenunsicherheit und Dichte des SLAM-Graphen gemacht werden. Die Abbildungen 6.16, 6.17, 6.18 und 6.15 stellen einen direkten Vergleich zwischen verschiedenen Einstellungen der Schwellwerte dar. Die Kartierung fand auf Basis des selben Datensatzes unter den selben Programmeinstellungen(mit Ausnahme dieser Schwellwerte) statt.



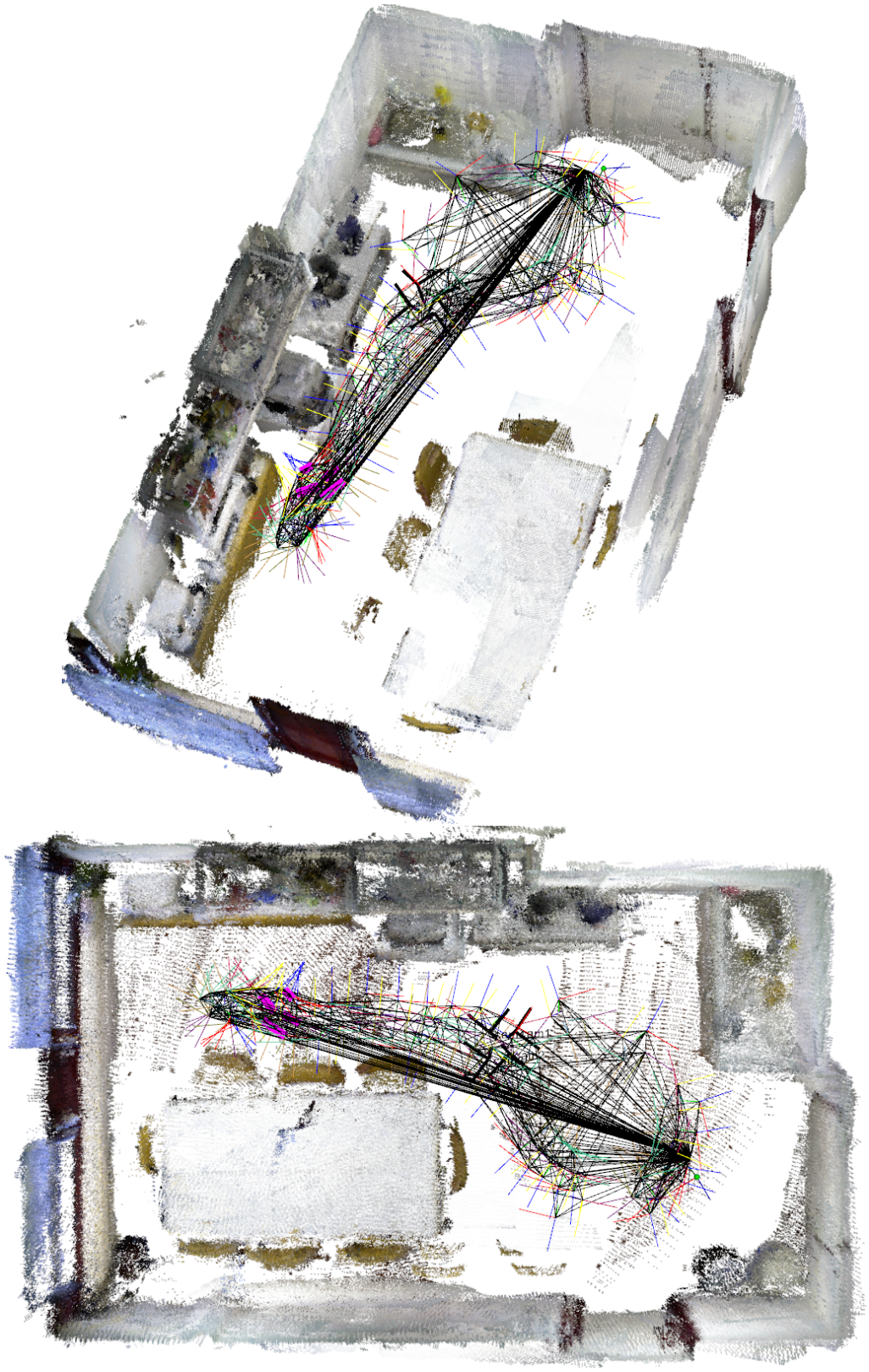
6.15: kein Pruning, Knotenzahl: 308, Kantenzahl: 1642, Speicherverbrauch: 3014MB



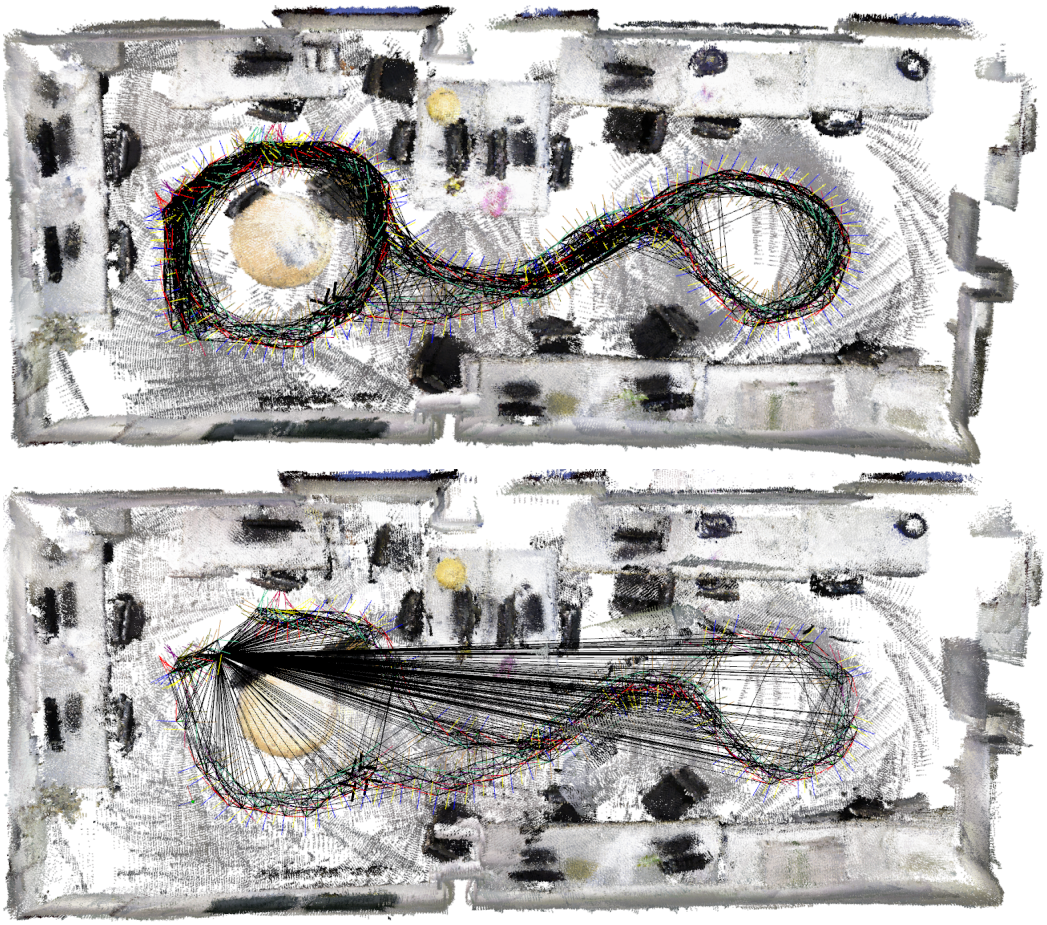
6.16: Varianzschwellwerte: (0,06; 0,04; 0,03; 0,03; 0,03; 0,03), Knotenzahl: 185, Kantenzahl: 990, Speicherverbrauch: 2207MB



6.17: Varianzschwellwerte: (0,075; 0,06; 0,04; 0,04; 0,04; 0,04), Knotenzahl: 145, Kantenzahl: 757, Speicherverbrauch: 2001MB



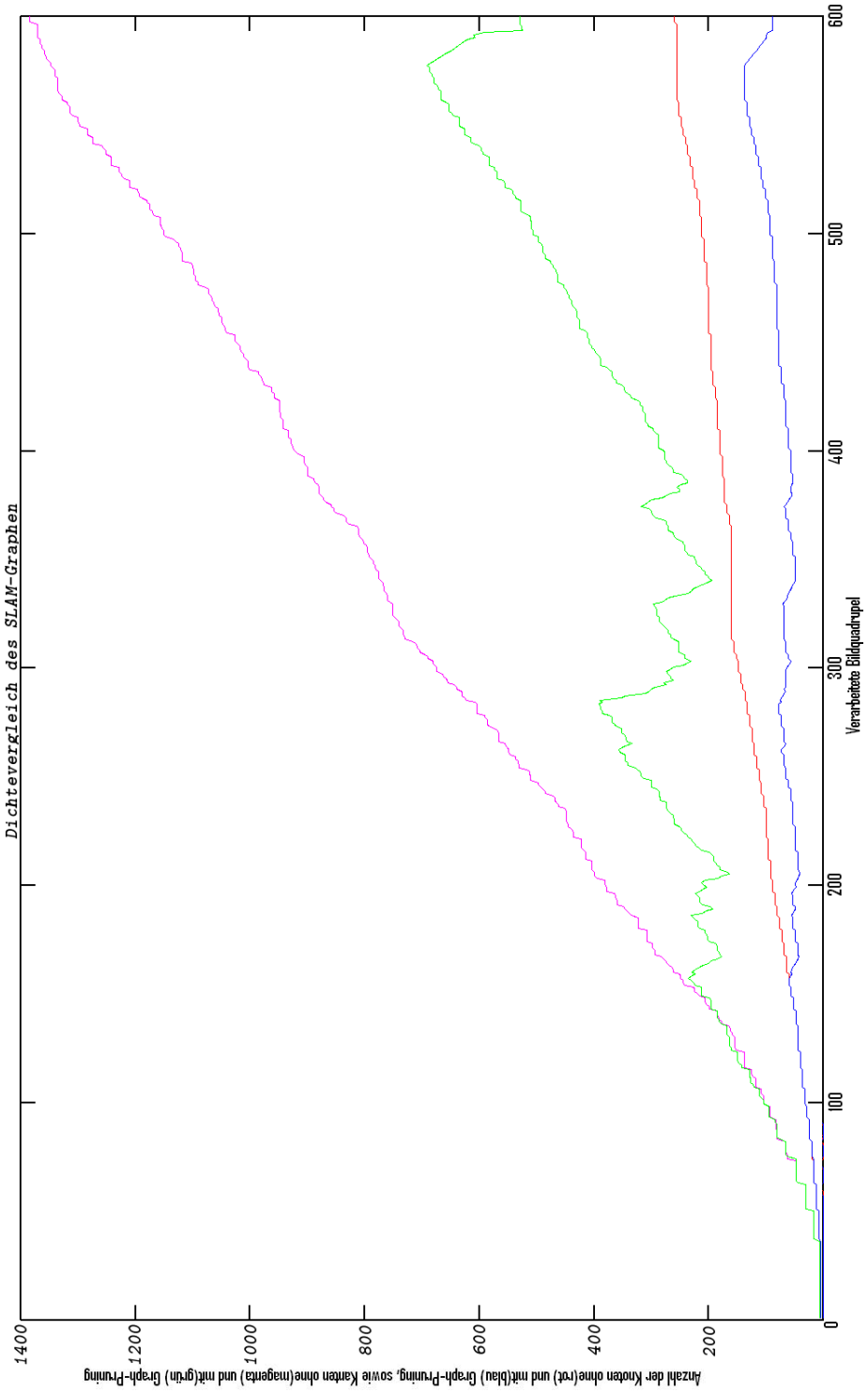
6.18: Varianzschwellwerte: (0,1; 0,08; 0,049; 0,049; 0,049; 0,049), Knotenzahl: 153, Kantenzahl: 918, Speicherverbrauch: 1851MB



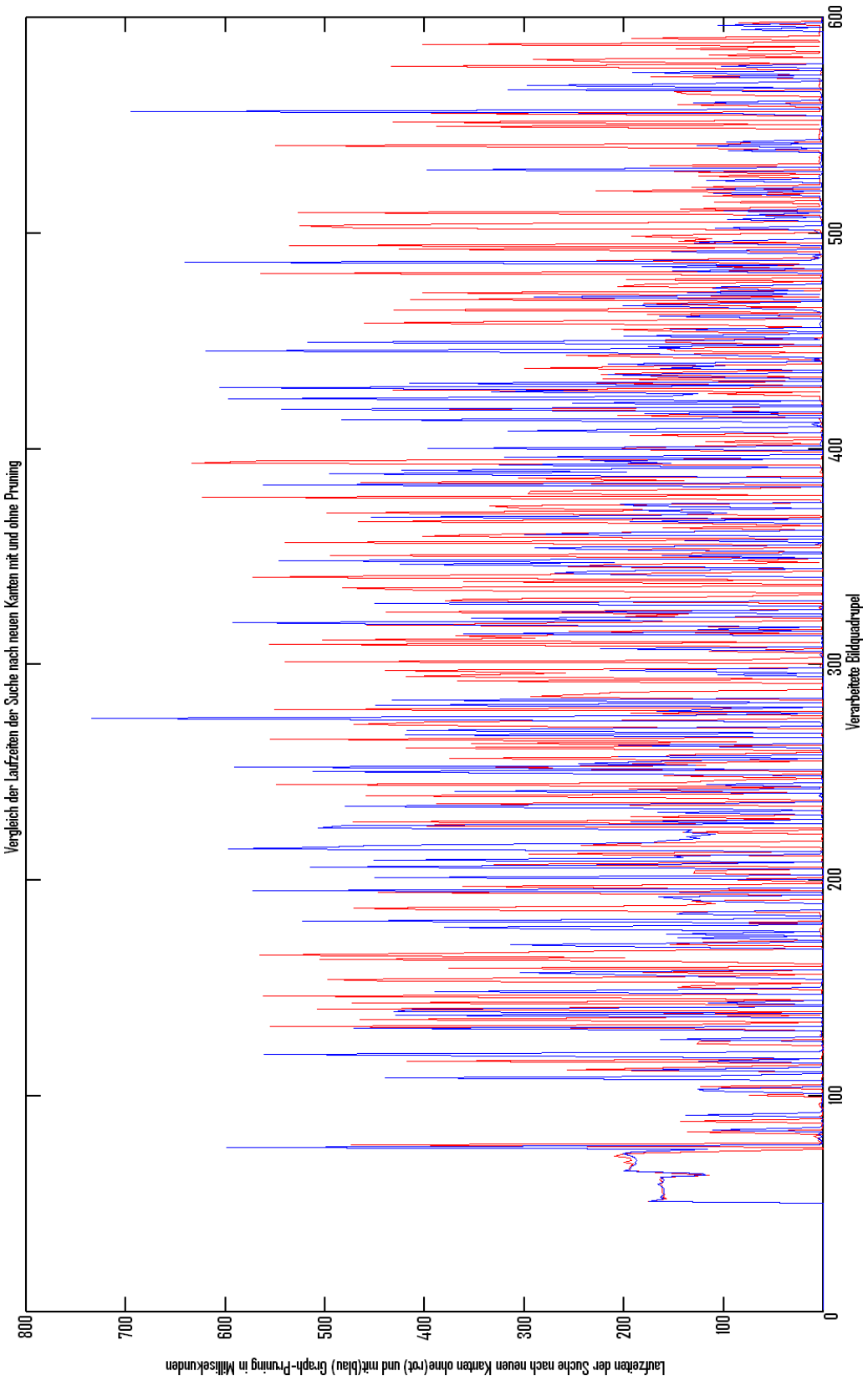
6.19: oben: kein Pruning, Knotenzahl: 628, Kantenzahl: 3321, Speicherverbrauch: 4849MB; unten: mit Pruning, Varianzschwellwerte: (0,08; 0,07; 0,06; 0,05; 0,05; 0,04), Knotenzahl: 365, Kantenzahl: 1934, Speicherverbrauch: 3330MB

Ein weiteres Beispiel in Abb 6.19 zeigt eine Kartierung des CI-Labors mit und ohne Pruning.

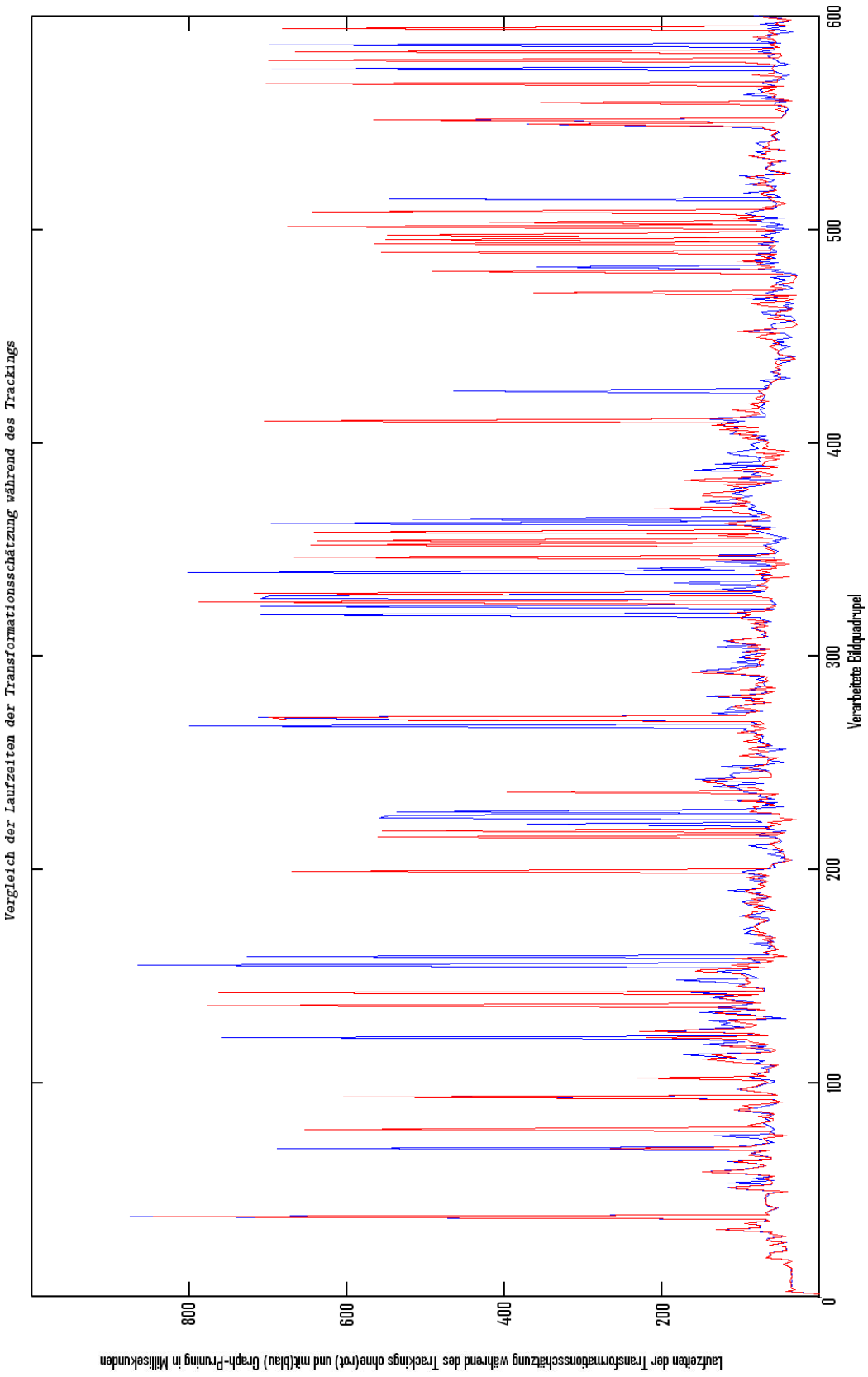
Um eine Übersicht der Laufzeiten der wichtigsten Schritte im Verfahren darzustellen, wurden diese zweimal in Folge auf einem Datensatz von der Abteilungsküche bis zum 600. verarbeiteten Bildquadrupel protokolliert. Dabei blieben die Programmeinstellungen in beiden Durchläufen unverändert, lediglich das Pruning war einmal ein- und einmal ausgeschaltet. Dokumentierende Plots dazu befinden sich in den Abbildungen 6.21, 6.24, 6.23, 6.22 und 6.20.



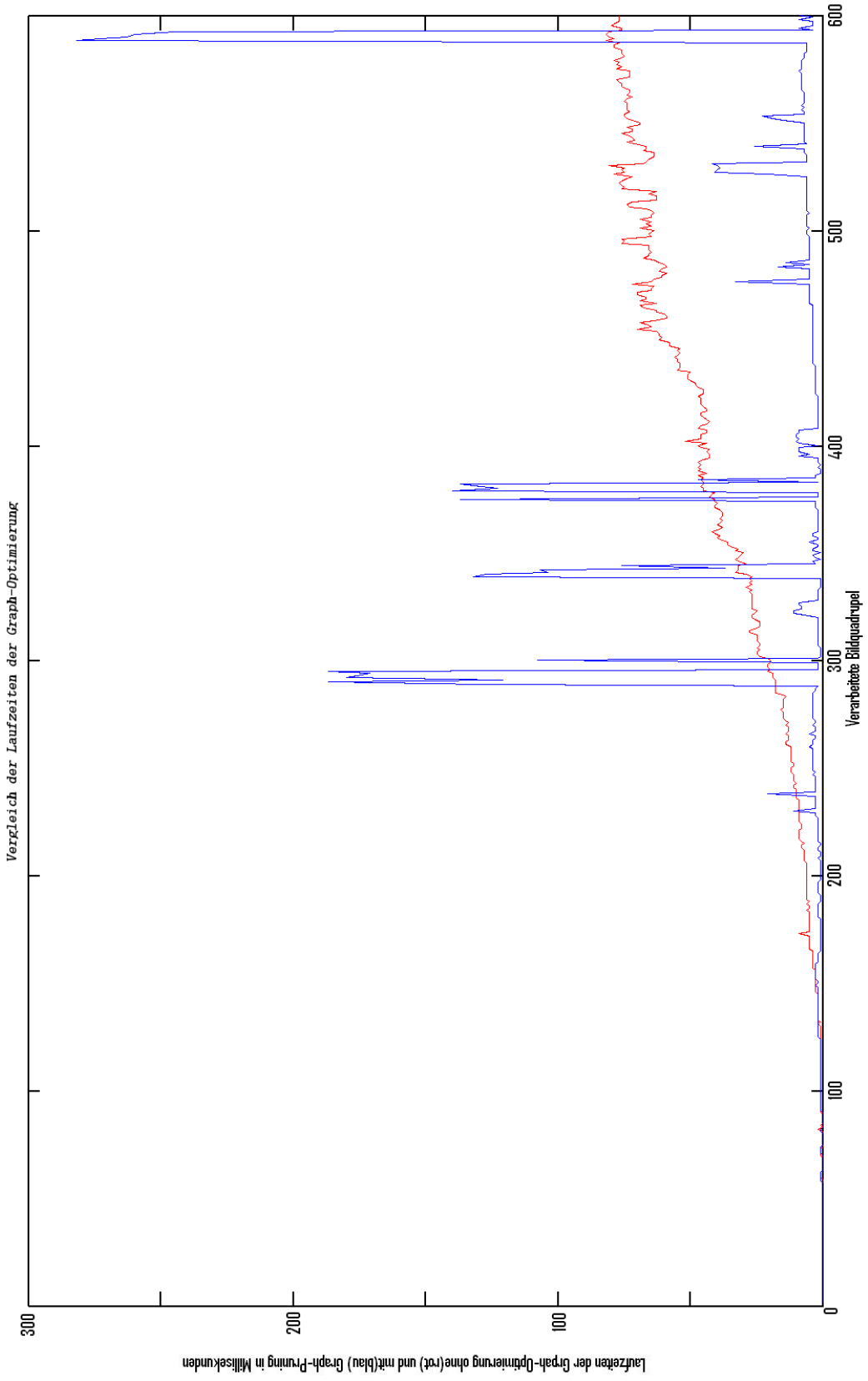
6.20: Entwicklung der Knoten und Kanten im Vergleich



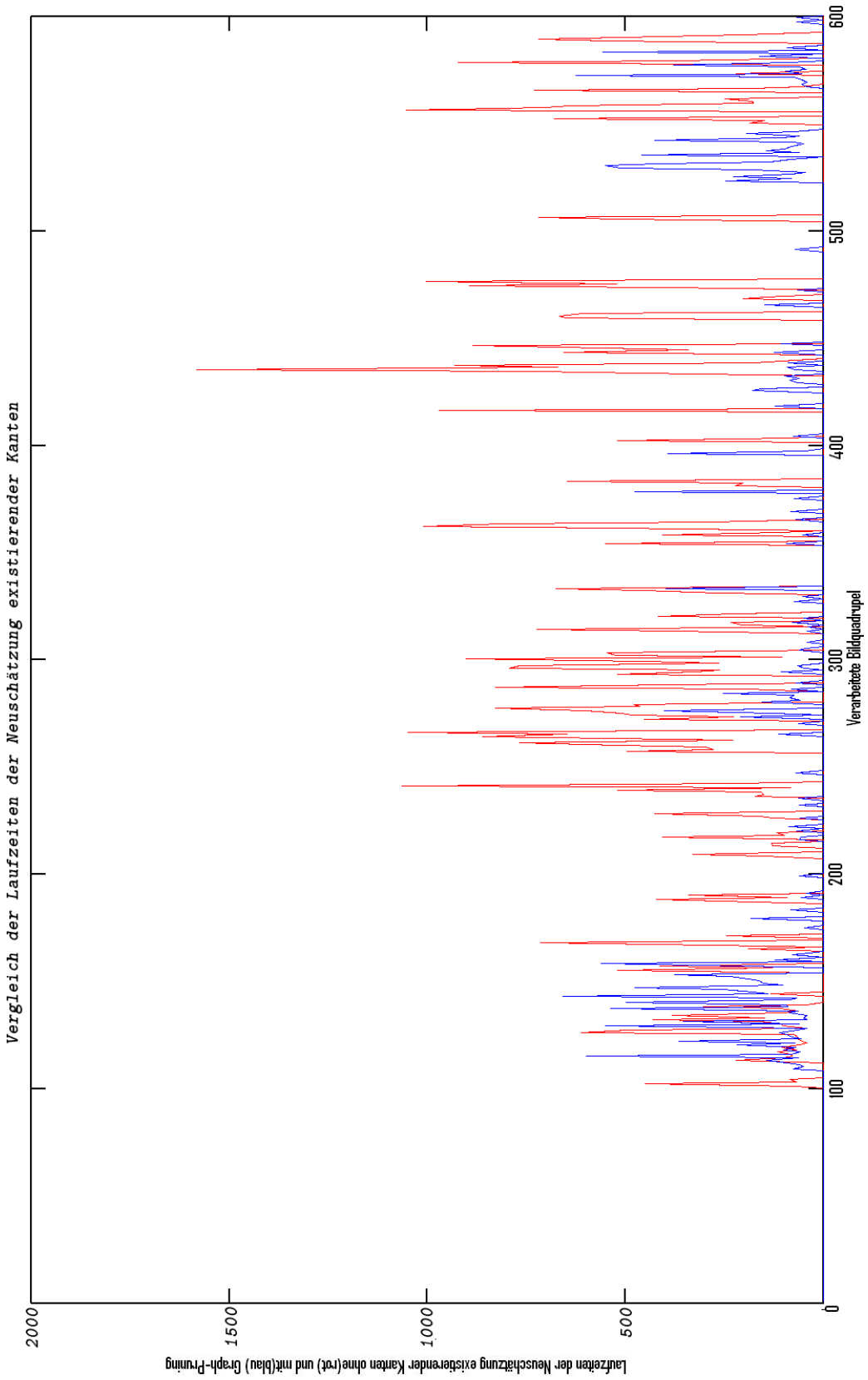
6.21: Laufzeiten der Suche nach neuen Kanten



6.22: Laufzeiten der Transformationsschätzung beim Tracking



6.23: Laufzeiten der Graph-Optimierung



6.24: Laufzeiten der Kantenneuschätzung

Zusammenfassung und Ausblick

Diese Arbeit zeigt, dass mit einem Multikamerasystem eine robuste Kartierung auch an den Stellen erreicht werden kann, an denen einige Kameras keine struktur- oder texturreichen Objekte im Blickfeld haben. Die, aus der Anordnung im 90° Versatz resultierende, Rundumsicht sorgt zudem dafür, dass ein Durchgang durch ein Gebiet bereits große Teile desselben erfasst. So müssen die Kameras keine Schwenkbewegungen ausführen um die Umgebung in weiten Teilen wahrzunehmen, wie es bei der Verwendung einer Kamera der Fall wäre. Schwächen hat das Verfahren auf dem aktuellen Stand noch, wenn Türdurchgänge stattfinden. Positive Registrierungen unterschiedlicher Wandteile aufeinander sorgen dafür, dass falsche Kanten in den Graphen einfliessen und an dieser Stelle Fehler entstehen. Die zusätzliche Verwendung von Punktmerkmalen(z.B. [21], [1] oder [29]) kann hier helfen, die Robustheit gegenüber Türdurchgängen zu erhöhen.
Weiterhin kam ein Graph-Pruning Verfahren zum Einsatz, welches Knotengruppen des SLAM-Graphen zur Laufzeit zusammenfasst und so für eine deutliche Ersparnis bezüglich des benötigten Speicherverbrauches sorgt. Gleichzeitig sorgen gut eingestellte Pruning-Parameter dafür, dass die die Kartenqualität nur sehr gering leidet. Das Pruning als solches ist noch rechenintensiv. Dem kann begegnet werden, indem eine Methode entworfen wird, die mehrere Multiresolution Surfel Maps zu einer Einzigen zusammenfassen kann, ohne dafür jeweils die Punktwolken, aus denen die Maps erstellt wurden, neu zu erzeugen.

Literaturverzeichnis

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

[2] A. Censi. An accurate closed-form estimate of icp's covariance. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3167–3172. IEEE, 2007.

[3] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14, oct 2008.

[4] C. K. Chow and Senior Member C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14:462–467, 1968.

[5] China Internet Information Center (CIIC). `http://german.china.org.cn/travel/txt/2012-02/16/content_24656453_27.htm`.

[6] Timothy A. Davis. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[7] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29:2007, 2007.

[8] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Reasearch*, 25:2006, 2006.

[9] C. Estrada, J. Neira, and J. D. Tardos. Hierarchical SLAM: real-time accurate mapping of large environments. *IEEE Transactions on Robotics*, 21(4):588–596, aug 2005.

[10] Gabriela Gallegos, Maxime Meilland, Patrick Rives, and Andrew I. Comport. Appearance-based slam relying on a hybrid laser/omnidirectional sensor. In *IROS*, pages 3005–3010. IEEE, 2010.

[11] S. Gilles. Robust description and matching of images, 1998.

[12] WAZ NewMedia GmbH and Co. KG. `http://www.derwesten.de/spiele/news/microsofts-kinect-schafft-es-ins-guinness-buch-der-rekorde-id4428517.html`.

[13] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgbd mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010.

[14] Christoph Hertzberg. A framework for sparse, non-linear least squares problems on manifolds, 2008.

[15] Dirk Hähnel, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic . . . In *IN PROC. OF THE IEEE/RSJ INT. CONF. ON INTELLIGENT ROBOTS AND SYSTEMS (IROS*, pages 206–211, 2003.

[16] Timor Kadir and Michael Brady. Saliency, scale and image description. *Int. J. Comput. Vision*, 45(2):83–105, nov 2001.

[17] Kurt Konolige and James Bowman. Towards lifelong visual maps. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, pages 1156–1163, Piscataway, NJ, USA, 2009. IEEE Press.

[18] H. Kretzschmar, C. Stachniss, and G. Grisetti. Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders. In *IROS*, San Francisco, CA, USA, 2011.

[19] H. Kretzschmar, C. Stachniss, and G. Grisetti. Pose graph compression for laser-based SLAM. In *Proc. of the Int. Symposium of Robotics Research (ISRR)*, Flagstaff, AZ, USA, 2011. Invited presentation.

[20] Manolis I. A. Lourakis and Antonis A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Trans. Math. Softw.*, 36(1):2:1–2:30, mar 2009.

[21] David G. Lowe. Distinctive image features from scale-invariant keypoints, 2003.

[22] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, 2004.

[23] MIA Group. `http://www.mia.uni-saarland.de/Teaching/MFI07/kap47.pdf`.

[24] P. Newman, D. Cole, and K. Ho. Outdoor slam using visual appearance and laser ranging. In *In IEEE International Conference on Robotics and Automation*, pages 1180–1187, 2006.

[25] Paul Newman and Kin Ho. Slam Ů loop closing with visually salient features, 2005.

[26] Kai Ni, Drew Steedly, and Frank Dellaert. Tectonic sam: exact, out-of-core, submap-based slam. In *In Proc. IEEE International Conference on Robotics and Automation*, pages 1678–1685, 2007.

[27] M. Paton. dynamic rgb-d mapping. 2011.

[28] Radu Bogdan Rusu, Eddie Cohen, Tomoto Shimizu Washio, Matt Bell, Eray Berger, Robert Wang. `www.openni.org`.

[29] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski. Orb: An efficient alternative to sift or surf. In *ICCV'11*, pages 2564–2571, 2011.

[30] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[31] Conrad Electronic SE. `http://www.conrad.de/ce/de/product/909469/?hk=SEM&insert_kz=VQ&WT.srch=1&scamp=PLA`.

[32] Aleksandr V. Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp.

[33] C. Stachniss and W. Burgard. Exploration with active loop-closing for fastslam. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[34] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[35] Jörg Stückler and Sven Behnke. Multi-resolution surfel maps for efficient dense 3d modeling and tracking. *Journal of Visual Communication and Image Representation*, 2013. To appear.

[36] Gerd Wiese. `http://www.landschaftsfotos.eu/name/zeitachse/jahr/2012/monat/oktober/fuer/21097/fotograf/gerd-wiese.html`.

[37] Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras. `http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/17-icp.pdf`.

Danksagung

Ich danke Jörg Stückler für die vielen Vorschläge, Hilfestellungen und Gespräche, die er als mein Ansprechpartner bei dieser Arbeit mit mir hatte. Ebenso danke ich Herrn Prof. Dr. Sven Behnke für die Betreuung dieser Arbeit sowie viele nützliche Anmerkungen und Vorgehensvorschläge, die er zu dieser Arbeit gab. Ich danke David Dröschel dafür, dass er mir insbesondere in der Einarbeitungsphase geduldig über viele Programmierfallen hinweg geholfen hat. Allgemein gilt mein Dank der gesamten Abteilung VI im Institut für Informatik, da ein jeder mich durch Anmerkungen, Vorschläge, Organisation(An dieser Stelle danke ich besonders Marion und Nikola aus dem Sekretariat.) oder unterhaltsame Auflockerung der Arbeit unterstützt hat. Ganz besonders geht auch mein Dank an Petra und Bernd Kiel, die mir auf privater Ebene eine große Menge an Unterstützung geleistet haben, sowie an alle Korrekturleser dieser Arbeit, die mir an manch einer Stelle die Augen geöffnet haben.

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und ausschliesslich unter Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Bonn, den 18.02.2013

Stephan Stroucken