UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Arbeitsgruppe Aktives Sehen

Implementierung von 6D SLAM auf Basis eines schnellen ICP-Algorithmus

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von

Peter Schneider

Betreuer:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	Dipl.-Inf. Johannes Pellenz, Institut für Computervisualistik, Fachbereich Informatik

Koblenz, im Oktober 2006

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den 10. 9. 2007 Unterschrift

Zusammenfassung

Große Gebiete lassen sich auf Grund von Schattenbildung und begrenzter Scanreichweite nicht mit einem einzigen 3D-Scan aufnehmen. Um konsistente dreidimensionale Karten dieses Gebietes zu erzeugen müssen also mehrere Scans zusammengefügt werden. Soll dieses Matchen der Scans automatisch geschehen, so kann es wegen fehlerhaften Translations- und Rotationsdaten, die die unterschiedlichen Positionen der Scans beschreiben, zu inkonsistenten Karten kommen. Um dies zu vermeiden wird in dieser Arbeit ein schneller Iterativ Closest Points Algorithmus [Nüc06] implementiert, der versucht, Fehler in diesen sechs Freiheitsgraden zu korrigieren. Das Verfahren soll im Rahmen dieser Arbeit in die schon vorhandene Software unseres Roboters eingebunden werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Will man mit einem 3D-Laserscanner größere Objekte scannen stößt man, wenn man nur einen einigen Scan verwendet, schnell an Grenzen. Diese Grenzen resultieren aus der eingeschränkten Reichweite des Scanners und den Schattenwürfen von sich verdeckenden Objekten. So lassen sich mit einem einzelnen 3D-Scan nur Objekte oder Bereiche scannen, die maximal die Größe der Reichweite des Laserscanners haben (siehe Bild 2.3). Sollen nun größere oder sich verdeckende Objekte aufgenommen werden, so müssen mehrer Scans aus unterschiedlichen Scan-Positionen gemacht werden. Diese einzelnen lokalen Scans müssen danach zu einem gesamten globalen Scan zusammengefügt werden. Um dies korrekt zu erreichen, muss die Verschiebung und Drehung der einzelnen lokalen Scans zueinander bekannt sein. Ein korrektes Zusammenfügen bedeutet hierbei, dass ein in der Welt von zwei unterschiedlichen lokalen Scans aufgenommener Punkt auch in dem, aus diesen beiden lokalen Scans zusammengefügten globalen Scan, als ein einzelner Punkt dargestellt wird. Um Scans aus verschiedenen Positionen aufzunehmen wurde der Scanner auf einen Roboter, im Folgenden „Robbie“ genannt, installiert. Dieser liefert Daten bezüglich Verschiebung und Drehung der einzelnen lokalen Scans die auf den Odometriedaten des Roboters beruhen. Allerdings sind diese Odometriedaten zum Einen nicht vollständig, einige Verschiebungs- und Drehungsdaten fehlen, zum Anderen durch das Rutschen der Roboterräder stark verfälscht. Daher muss ein Korrekturalgorithmus angewand werden. Da wir mit dreidimensionalen Scans arbeiten, muss dieser sowohl die Verschiebung in den

drei Koordinatenachsen X, Y und Z als auch die Drehung um diese drei Achsen beachten. Es handelt sich also um sechs Freiheitsgrade die korrigiert werden müssen.

Um dieses sechs-dimensionale simultanius localising and mapping (kurz SLAM) Problem zu lösen, wird in dieser Studienarbeit ein itterativ closest point (kurz ICP) Algorithmus beschrieben und implementiert. Dieser Algorithmus basiert auf dem von Nüchter beschriebenen 6D-SLAM ICP-Algorithmus in [Nüc06]. Die für diese Studienarbeit verwendeten 3D-Scans werden, anders als bei Nüchter, von einem umgebauten Hokuyo Scanner erzeugt, beschrieben von Pellenz in [PDMP06].

Da dieser Scanner eine sehr viel geringere Reichweite hat, dabei allerdings eine höhere Auflösung besitzt, muss ein neues Verfahren zur Punktreduktion entwickelt werden und der eigentliche ICP-Algorithmus auf diese geänderten Scanwerte angepasst werden.

Der Aufbau der Arbeit ist wie folgt:

Im Anschluss findet sich eine nähere Beschreibung des verwendeten 3D-Laserscanners sowie des Dateiformates in dem die einzelnen Scans abgespeichert werden.

In Kapitel 3 wird dann der ICP-Algorithmus mit allen einzelnen verwendeten Bestandteilen, wie dem Algorithmus zur Punktreduktion und dem Verfahren zur schnellen Punkpaarsuche, beschrieben.

Dem folgt in Kapitel 4 eine Beschreibung der durchgeführten Experimente sowie deren Ergebnisse.

Abschließend werden in Kapitel 5 Ergebnisse zusammengefasst und soweit möglich bewertet. Außerdem wird ein Ausblick auf weitere Verwendungsmöglichkeiten dieses Verfahrens gegeben.

Kapitel 2

Robbies bisherige Scantechnik

Hier soll die bisher auf unserem Roboter verwendete Scantechnik, speziell der verwendete 3D-Scanner, sowie das von uns entwickelte Format zum Abspeichern der einzelnen Scans beschrieben werden. Außerdem wird ein Einblick in die Softwarearchitektur von Robbie gegeben, da auch die Implementierung des ICP-Algorithmus in dieses bestehende Programm Teil dieser Studienarbeit ist.

2.1 Technische Beschreibung des 3D-Scanner

Bei dem in dieser Studienarbeit verwendeten Scanner handelt es sich um einen umgebauten Laser Range Finder der Firma Hokuyo vom Typ URG-04LX. Dieser Scanner hat folgende Eigenschaften:

- Öffnungswinkel: 240°
- Scanschritte pro Gesamtscan: 682
- Winkelauflösung: 0,36°
- Reichweite: 20mm - 4000mm (in der hier verwendeten Konfiguration)
- Benötigte Zeit für einen Scan: 100 msec (daraus ergeben sich 10 Scans pro Sekunde)

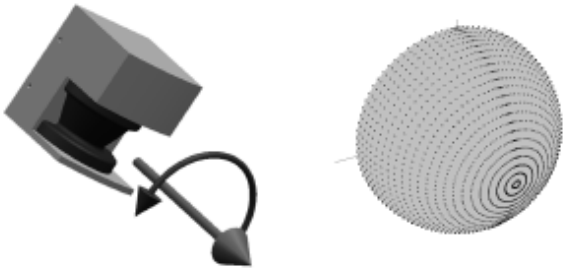

Bild 2.1: Hier ist links die Scanbewegung, rechts der abgedeckte Scanbereich zu sehen. Quelle: [WW03] Seite 2

Er liefert somit eine zweidimensionale Abtastung eines Raumes, also exakt eine Schnittebene. Um mit diesem Scanner dreidimensionale Scans generieren zu können, wird dieser auf einen Servomotor installiert. Der Scanner wird auf diesem so positioniert, dass er um eine Achse, die dem mittleren Scanstrahl entspricht, also exakt dem Laserstrahl an Position 341, um 180° in mehreren Schritten dreht (siehe Bild 2.1). Die Anzahl der einzelnen zweidimensionalen Scanschritte während einer 180° Drehung ist dabei variabel, sollte aber nicht unter 50 liegen, da der 3D-Scan sonst eine zu geringe Auflösung besitzt. Die Auflösung von einhundert Scanschritten hat sich während der Arbeit mit dem Scanner als praktischste Auflösung erwiesen. Sie bietet eine gute Punktauflösung, wobei aber die Datenmenge pro einzelnem Scan überschaubar bleibt (200-300 kB). Mit der so aufgenommenen Serie von zweidimensionalen Scans und den jeweils zu ihnen gehörenden Winkeln lässt sich eine dreidimensionale Punktewolke erstellen, die dem dreidimensionalen Kugelraum im Radius der Laserscannerreichweite entspricht. Zu beachten ist dabei der kegelartige tote Winkel des 3D-Scans, der durch den gegebenen Öffnungswinkel des Laserscanners entsteht. Man erhält pro 3D-Scan maximal 68200 Punkte, falls im gesamten gescannten Bereich Objekte innerhalb der Reichweite des Scanners liegen. An den Stellen, an denen keine Objekte in Reichweite liegen, werden keine Punkte dargestellt. Der Scanner liefert für diese Positionen im Scan den Fehlercode „19“ zurück. Solche Positionen werden dann bei der Umrechnung zur Punkwolke nicht mehr berücksichtigt. Näheres zum Aufbau des 3D-Scanners ist in der Studienarbeit [Del07] von Delis nachzulesen.

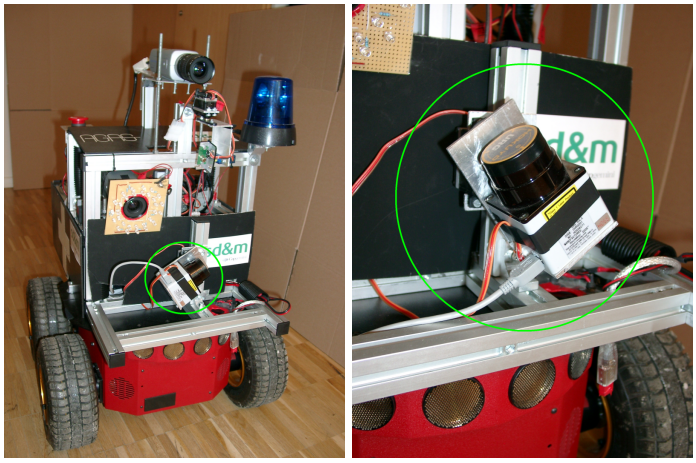

Bild 2.2: Beide Bilder zeigen den auf Robbie installierten 3D-Scanner (umkreist)

2.2 Beschreibung der erzeugten Daten

Um den Speicherbedarf der einzelnen Scans möglichst gering zu halten, wurde im Laufe der Arbeit mit dem 3D-Scanner ein neues Dateiformat (.l3d) spezifiziert, in dem die dreidimensionalen Laserscans abgespeichert werden. Dies nutzt zur effektiven Speicherung ein Kugelkoordinatensystem. Da der 3D-Scanner während eines Scanvorgangs nicht weiterbewegt werden darf, ist das Kugelkoordinatensystem hierbei besonders praktisch, da der 3D-Scanner die Daten ohnehin in dieser Art Koordinatensystem liefert. Im Folgenden soll auf die Umrechnung von Kugel- zu karthesischen Koordinaten sowie auf das l3d-Dateiformat eingegangen werden.

2.2.1 Von Kugelkoordinaten zu karthesischen Koordinaten

Beim Kugelkoordinatensystem handelt es sich um ein räumliches Polarkoordinatensystem. Die Punkte des Raums werden also mit Hilfe von zwei Winkeln und dem Abstand zum Mittelpunkt angegeben. In Bild 2.4 wird ein solches Kugelkoordinatensystem schematisch dargestellt. Der Punkt P wird hierbei durch die beiden Winkel θ und ϕ sowie seinen Abstand zum Mittelpunkt und somit zum Ursprung des Koordinatensystems $\boldsymbol{r}$ an-

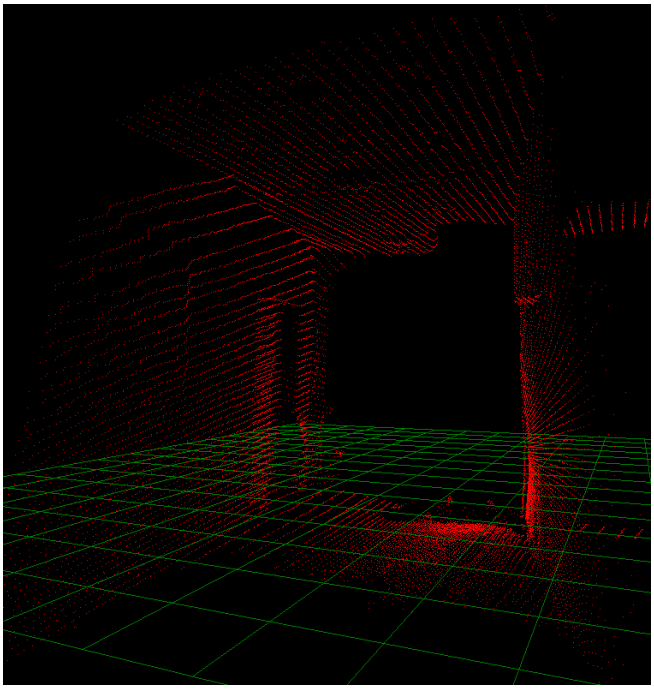

Bild 2.3: Einzelner 3D-Scan eines Flures

gegeben. Somit wäre bei unendlich hoher Auflösung aller Winkel jeder Punkt im Raum definiert. Die Umrechnung in karthesische Koordinaten geschieht durch folgende drei Formeln:

$$\mathsf{P}_{\mathrm{x}} = \boldsymbol{r} \cdot \sin\theta \cdot \cos\phi \tag{2.1}$$

$$\mathsf{P}_{\mathrm{y}} = \boldsymbol{r} \cdot \sin\theta \cdot \sin\phi \tag{2.2}$$

$$\mathsf{P}_{\mathrm{z}} = \boldsymbol{r} \cdot \cos\theta \tag{2.3}$$

Diese Formeln gelten wie aufgeführt, wenn, wie in Bild 2.4, die Drehachse der z-Achse des karthesischen Koordinatensystems entspricht. In unserer Scannerkonstruktion ist das dann der Fall, wenn der Scanner mit Blickrichtung nach oben auf Robbie installiert ist. In anderen Fällen, die Drehachse zeigt also im karthesischen System entlang einer anderen Achse, müssen 2.1, 2.2 und 2.3 entsprechend abgeändert werden.

Die Verwendung des Kugelkoordinatensystems ist deshalb angebracht, weil der 3D-Scanner die Daten in ähnlichem Format liefert, als Folge von Abstandswerten des Gegenstandes

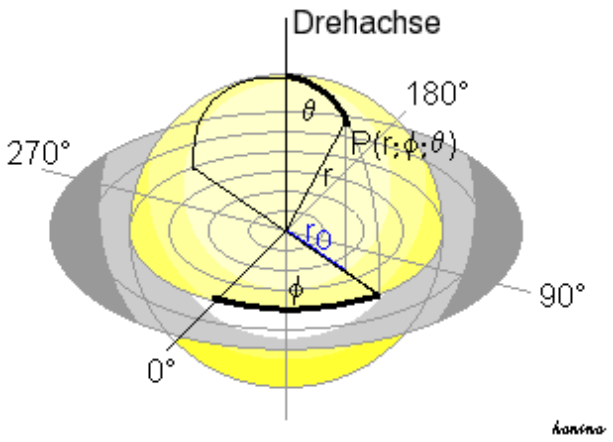

Bild 2.4: Schema eines Kugelkoordinatensystems, Quelle [Wik06]

zum Scanner sowie die zu den jeweiligen Werten gehörenden beiden Winkeln. Der Scanner liefert mit jedem einzelnen 2D-Scan, der im Zuge eines 3D-Scans gemacht wird, bei einer festen Winkelauflösung immer 682 Abstandswerte. Daraus lässt sich innerhalb jedes 2D-Scans der zur Umrechnug benötigte Winkel θ ermitteln. Ebenso verhält es sich bei der Drehung des gesamten Scanners während eines 3D-Scans um seine Drehachse. Diese Drehung wird in einer vorher angegebenen Anzahl von Schritten n vollzogen. Zwischen zwei dieser Schritte wird dann ein 2D-Scan geschossen, der der Serie von 2D-Scans, aus denen sich ein 3D-Scan zusammensetzt, hinzugefügt wird. Da die Größe der Schritte während eines solchen 3D-Scans immer gleich ist, nämlich $\frac{n}{180°}$, lässt sich daraus auch der Winkel ϕ berechnen. Dieser gilt dann jeweils für einen gesamten 2D-Scan. Für unsere Umrechnung benötigen wir pro 3D-Scan also nur ϕ_{start}, $\phi_{Reichweite}$ sowie die Anzahl der Schritte. Die Winkelangabe der Reichweite drückt dabei aus, um wieviel Grad sich der Scanners dreht, startend bei ϕ_{Start}. Um eine Drehung entgegen des Uhrzeigersinns zu bekommen, wird diese Angabe also negativ. Die Winkel θ_{start}, $\theta_{Reichweite}$ sowie die 682 Abtastschritte innerhalb eines 2D-Scans bleiben immer konstant und müssten daher nicht mehr angegeben werden, werden aber Zwecks größerer Flexibilität trotzdem zugefügt.

Allerdings sind genau diese innerhalb eines einzelnen lokalen 3D-Scans festgelegten Werte auch der Grund, wieso sich das Kugelkoordinatensystem mit dieser Art von Formatie-

rung nicht eignet, um einen globalen und aus mehreren lokalen Scans zusammengefügten Scan zu halten. Bei einer zu Beginn festgelegten Winkelauflösung des Scans würde in größerer Entfernung zum Ursprung die Punktauflösung zu gering werden. Somit wird das Kugelkoordinatensystem nur für die lokalen Scans verwendet, der globale wird in karthesischen Koordinaten gehalten.

2.2.2 Aufbau des l3d-Dateiformats

Zum Abspeichern, beziehungsweise generell zum Übertragen einzelner lokaler Scans, werden die oben beschriebenen Eigenschaften des Kugelkoordinatensystems ausgenutzt, um eine möglichst geringe Datengröße zu erreichen. Da der Winkel $\Delta\phi$ zwischen den einzelnen 2D-Scans sowie der Winkel $\Delta\theta$ zwischen den einzelnen Abtastwerten eines 2D-Scans innerhalb eines 3D-Scans gleich sind, genügt es theoretisch also, nur diese beiden Δ-Winkel zu übertragen, um die Winkelauflösung eines Kugelkoordinatensystems zu erhalten. Um nun aber eine konsistente Umrechnung in karthesische Koordinaten bei variablen Start- und End-Winkeln zu ermöglichen, werden statt dessen ϕ_{Start}, $\phi_{Reichweite}$, θ_{Start}, $\theta_{Reichweite}$ sowie die Auflösung, also Schrittanzahl, zwischen den jeweils zusammengehörenden Winkelpaaren gespeichert. Daraus lassen sich beide Δ-Winkel errechnen. Für zusätzliche Flexibilität sorgt des Weiteren die Übertragung der Richtung der Drehachse. Somit können Scans, die mit unterschiedlichen Scannerblickrichtungen aufgenommen wurden, in das gleiche karthesische Koordinatensystem übertragen werden.

Der Aufbau einer l3d-Datei ist nun wie folgt:

1. Datenformat (Kugelkoordinaten oder karthesische Koordinaten)
2. Beschreibung des polaren Scanbereichs ($\theta_{Start}, \theta_{Reichweite}$, Scanschritte)
3. Beschreibun des azimutalen Scanbereichs ($\phi_{start}, \phi_{Reichweite}$, Scanschritte)
4. Beschreibung der Drehachse
5. Scandaten (Folge von Abstandswerten, die binär gespeichert werden)

Um in l3d-Dateien auch die globalen Scans in karthesischen Koordinaten abspeichern zu können, wurde das Dateiformat um dieses Koordinatensystem erweitert. Um welches Datenformat es sich dabei bei einer l3d-Datei handelt, ergibt sich aus dem Schlüssel „Datenformat“ zu Beginn der Datei. Handelt es sich um karthesische Daten, so ist der Aufbau der Datei nach Angabe des Datenformates wie folgt:

2. Anzahl der enthaltenen Punkte
3. Scandaten (binär gespeichert)

Die genaue Spezifikation befindet sich unter Anhang A.

2.3 Robbies Softwarearchitektur

Bei der Robbie-Software handelt es sich um eine modular aufgebaute Softwarearchitektur, wobei die einzelnen Module jeweils als unabhängiger Thread ablaufen. Sie kommunizieren untereinander mit Hilfe von Nachrichten, die die benötigten oder ermittelten Daten enthalten. Diese Nachrichten-Kommunikation wird vom Softwarekern verwaltet und gesteuert. Jedes Modul kann mehrere „Devices“ und Instanzen von „Workern“ halten. Hierbei handelt es sich um Programme, die die eigentliche Datenermittlung und -berechnung übernehmen. Die Module kümmern sich dabei nur um die Steuerung des Datenflusses. Im Speziellen handelt es sich bei Devices um Programme, die eine Schnittstelle zu einer Hardwarekomponente liefern und diese ansteuern beziehungsweise Daten von ihr abfragen können. Von diesen Devices kann immer nur eine Instanz existieren. Sie kümmern sich außerdem mit Hilfe eines Mutex darum, dass es nicht zu Konflikten bei der Ansteuerung der Hardwarekomponenten kommt. Ein Beispiel für so ein Device ist unser 3D-Laserscanner. Im Gegensatz dazu handelt es sich bei Workern um Programme, die reine Datenverrechnung übernehmen. Von ihnen kann es beliebig viele Instanzen geben. Ein Beispiel dafür wäre das im Zuge dieser Arbeit implementierte ICP-Programm, das die verfügbaren Scans matcht. All diese beschriebenen Elemente sind auch im Schema Bild 2.5 zu sehen. Da die zu ladenden Module beim Start des Robbie-Programmes separat und flexibel gewählt werden können, ist es möglich, das Gesamtprogramm soweit zu minimieren, dass nur die Tei-

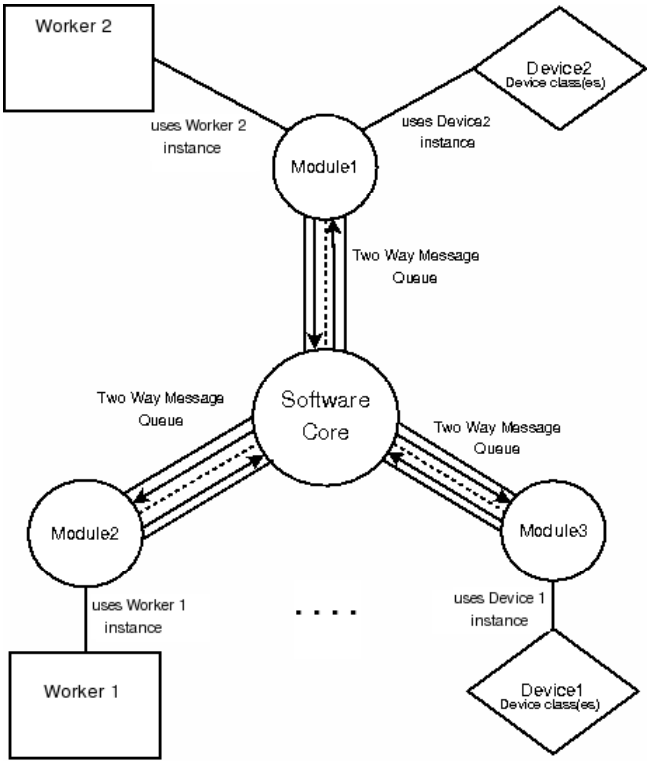

Bild 2.5: Schematische Darstellung der Robbies Softwarearchitektur

le gestartet werden, die man auch tatsächlich benötigt. So läßt sich das Robbie-Programm wahlweise auch nur mit dem in dieser Arbeit implementierten ICP-Modul starten. Durch das Fehlen der anderen Module werden dabei Ressourcen geschont, Funktionalitäten, für die das ICP-Modul auf andere Module zurückgreift sind dann allerdings nicht verfügbar. Eine nähere Beschreibung des ICP-Moduls mit samt seiner Abhängigkeiten findet sich im folgenden Kapitel.

Kapitel 3

Das ICP-Verfahren

In diesem Kapitel soll als Erstes auf die Grundidee und die verschiedenen Grundtypen von ICP-Algorithmen eingegangen werden, wobei kurz deren Eigenschaften und Leistungen erleutert werden. Anschließend werden die mathematischen Grundlagen und ein mathematischer Lösungsansatz erklärt. Darauf folgt die Beschreibung der Implementierung wobei die einzelnen Schritte im Programmablauf und ihre Abhängigkeiten erläutert werden.

3.1 Grundidee des ICP-Algorithmus

Zum besseren Verständnis der einzelnen Grundtypen soll zunächst die generelle Idee, die hinter dem Iterativ Closest Point Algorithmus steht, erleutert werden. Hierbei handelt es sich um einen Algorithmus, bei dem die Korrekturberechnung iterativ durchgeführt wird. Die Korrekturberechnung bezieht sich dabei auf die schon vorhandenen groben Verschiebungsdaten zwischen, in unserem Fall der Position eines ersten 3D-Scans und der Position eines zweiten, der mit dem ersten zusammengefügt werden soll. Die vorhandenen Verschiebungsdaten, im Folgenden initiale Daten genannt, können dabei z.B. aus Schätzungen, (fehlerbehafteten) Messungen oder Benutzereingaben stammen. Diese initialen Daten werden dann mit jeder einzelnen Iteration korrigiert und nähern sich somit der tatsächlich in der Welt durchgeführten Verschiebung an. Im Einzelnen lauten die Schritte:

1. Registrierung der beiden Scans,
2. Ermittelung der Punktpaare,
3. Berechnung der Korrektur.

Beim Start dieser Schleife werden zur Registrierung der Scans die initialen Daten verwendet. Beim Ermitteln der Punktkorrespondenzen wird dann für jeden Punkt aus dem einen Scan ein nächstgelegener Punkt aus dem anderen Scan gesucht, weil angenommen wird, dass diese beiden Punkte in der Welt den selben Ausgangspunkt haben. All diese Punktpaarungen werden dann zur Berechnung der Korrektur verwendet. Mit den dadurch ermittelten Korrekturdaten wird dann wieder eine Registrierung der beiden Scans durchgeführt und die Schleife beginnt von vorne. Dieser Ablauf wird so lange wiederholt, bis der in der Scanregistrierung gemessene Fehler zwischen beiden Scans einen Fehlergrenzwert unterschreitet.

3.2 Verschiedene ICP-Verfahren

Bei dem ICP-Algorithmus handelt es sich generell um ein iteratives Verfahren, bei dem Punktkorrespondenzen verwendet werden. Diese Punktpaare ergeben sich dabei jeweils aus einem Punkt des einen Scans und des dazu nächstgelegenen Punktes aus dem anderen Scan. Bei der Ermittlung dieser Punktkorrespondenzen lassen sich allerdings unterschiedlichste Verfahren anwenden, die auch zu unterschiedlichsten Leistungen in Geschwindigkeit und Matching-Qualität führen, wie Rusinkiewicz in [RL01] ermittelte. Da das Ermitteln der Punktpaarungen, die für die Berechnung verwendet werden, in jeder Itteration erneut vorkommt und im Vergleich zur darauf folgenden Berechnung der Korrektur sehr viel Zeit in Anspruch nimmt, liegt hier, also bei dem Schritt der Korrespondenzbestimmung, das Hauptmerkmal zur Unterscheidung verschiedener ICP-Verfahren. Unter anderem gibt es hierbei folgende Arten:

1. Suche zu jedem Punkt des einen Scans innerhalb einer maximalen Umgebung den zu ihm nächstgelegenen Punkt des anderen Scans.

2. Nimm für jeden Punkt des einen Scans den Punkt als Partner, der von dem Strahl, den wir von seiner Position aus in Richtung seiner Normalen schießen, getroffen wird. Die Normale eines Punktes wird hierbei mit Hilfe seiner direkten Punktnachbarn bestimmt.

3. Projeziere jeden Punkt des ersten Scans auf das Punktgitter des zweiten Scans aus Sicht der Aufnahmeposition des zweiten Scans.

4. All diese vorangegangenen Korrespondenzbestimmungsverfahren können mit Hilfe eines kD-Trees beschleunigt werden. Auch kann die Kompatibilität der Punktpaarungen beachtet werden in dem man nur Punktpaare weiterverwendet, deren Normalen ähnlich sind.

(Orginalauflistung siehe [RL01] Seite 3f)

Rusinkiewicz hat in [RL01] all diese ICP-Verfahren gegenübergestellt und mit Hilfe von Testscans unterschiedlichster Art ihre Leistungsfähigkeit ermittelt. Dabei wurde festgestellt, dass sich die Verfahren sowohl in Geschwindigkeit als auch in Qualität des Ergebnisses stark unterschieden. So konvergierte das ICP-Verfahren Nummer drei mit am schnellsten auf allen Testdaten, lieferte dabei aber auf einigen Testdaten gar keine richtigen Endergebnisse. ICP-Verfahren Nummer zwei war von Seiten der Geschwindigkeit her betrachtet meist im Mittelfeld, lieferte allerdings ebenfalls nicht immer korrekte Ergebnisse, wobei er im Fall eines falschen Ergebnisses zum Konvergieren am längsten benötigte. Die zumeist langsamsten aber auch robustesten Ergebnisse lieferte Verfahren Nummer eins. Mit diesem gelang es Rusinkiewicz, alle Testdaten korrekt zu matchen (siehe Bild 3.1). Er kam somit zu dem Schluss, dass diese Verfahren die beste Wahl für alle Arten von zu matchenden Scandaten ist (siehe [RL01] Seite 4 Zeile 38ff).

Weil der in dieser Arbeit zu implementierende ICP-Algorithmus auf allen Arten von Scandaten laufen soll, fiel die Entscheidung, ausgehend von den Vergleichen von Rusinkiewicz, zugunsten des Verfahrens Nummer eins, beschleunigt mit Hilfe eines kD-Trees.

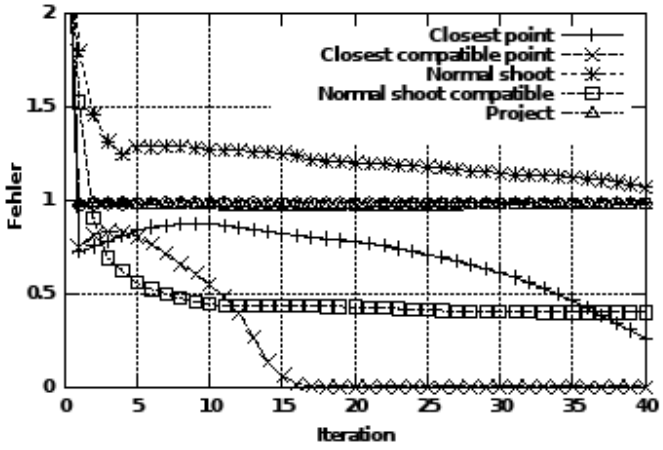

Bild 3.1: Diese Kurven zeigen die Fehlerkorrektur der einzelnen ICP-Verfahren bei schwersten Testdaten (Quelle: [RL01])

3.3 Mathematische Lösung

Für die mathematische Lösung der Korrekturwerte gibt es mehrere Ansätze, unter anderem die mit Hilfe von Quaternionen, Dualquaternionen oder Orthonormal-Matrizen. Im Programm implementiert sind die Lösung mit Hilfe der Singulärwertzerlegung sowie die Lösung mit Hilfe einer Punkt-zu-Punkt-Abschätzung, wobei hier nur auf die Lösung mit Hilfe der Singulärwertzerlegung näher eingegangen werden soll. Auf die genaue mathematische Herleitung sowie den Beweis der Konvergenz wird allerdings verzichten, da diese in diversen Arbeiten (Titel bei den jeweiligen folgenden Ansätzen angegeben) erfolgt ist.

Für alle diese Lösungsansätze gelten dabei folgende Voraussetzungen:

- Aus den vorher erstellten Punktpaarungen besitzen wir zwei Punktemengen, eine Menge M aus dem Modellscan, also dem globalen Scan, und eine Menge D aus dem Datenscan, dem lokalen Scan.
- Es gilt: $\forall \mathsf{m} \in M \dot{\exists} \mathsf{d} \in D$
  Somit ist die Anzahl N der Elemente in beiden Mengen gleich.
- Der Punkt $\mathsf{d_k} \in D$ ist Partner von $\mathsf{m_i} \in M$ wenn $i = k$

Außerdem werden folgende Berechnungen im Voraus gemacht:

$$\mathsf{c_m} = \frac{1}{N}\sum_{i=1}^{N} m_i, \qquad \mathsf{c_d} = \frac{1}{N}\sum_{i=1}^{N} d_i \tag{3.1}$$

Mit 3.1 wird das jeweilige Zentrum der beiden Punktemengen ermittelt. Mit Hilfe ihrer Zentren ergeben sich folgende Punktemengen:

$$M' = \{\mathsf{m'_i} = \mathsf{m_i} - \mathsf{c_m}\}_{1,\ldots,N}, \qquad D' = \{\mathsf{d'_i} = \mathsf{d_i} - \mathsf{c_d}\}_{1,\ldots,N} \tag{3.2}$$

Die Transformation zwischen den beiden Punktemengen wird nun auf in eine Translation $\boldsymbol{t}$ und eine Rotation $\boldsymbol{R}$ aufgeteilt. Um die Rotation zu errechnen, muss folgende Fehlerfunktion minimiert werden:

$$\mathrm{E}(\boldsymbol{R}) = \sum_{i=1}^{N} \|\mathsf{m'_i} - \boldsymbol{R}\mathsf{d'_i}\|^2 \tag{3.3}$$

(Herleitung von 3.3 in [Nüc06] Kapitel 3.1.1)

Die Translation ergibt sich dann aus $\boldsymbol{t} = \mathsf{c_m} - \boldsymbol{R}\mathsf{c_d}$.

Die Minimierung des Fehlers der Gleichung 3.3 mit Hilfe einer Singulärwertzerlegung wurde von Arun, Huang und Blostein in [AHB87] entwickelt. Hierbei wird zuerst die Rotation anhand einer orthonormalen 3×3 Matrix ermittelt, darauf erfolg dann die Berechnung der Translation anhand des rotierten Zentrums der Datenpunktmenge D. Die Rotationsmatrix $\boldsymbol{R}$ ergibt sich dabei aus $\boldsymbol{R} = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{V}$ Die für diese Formel benötigten Matrizen $\boldsymbol{U}$ und $\boldsymbol{V}$ stammen aus der Singulärwertzerlegung $\boldsymbol{H} = \boldsymbol{U}\Sigma\boldsymbol{V}^{\mathrm{T}}$ der Korrelationsmatrix $\boldsymbol{H}$. Diese 3×3 große Korrelationsmatrix wird folgendermaßen belegt:

$$\boldsymbol{H} = \sum_{i=1}^{N} \mathsf{d'_i}\mathsf{m'^{T}_i} = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}, \tag{3.4}$$

mit $S_{xx} = \sum_{i=1}^{N} \mathsf{m'_{ix}}\mathsf{d'_{ix}}$, $S_{xy} = \sum_{i=1}^{N} \mathsf{m'_{ix}}\mathsf{d'_{iy}}$, ... ($\mathsf{m'_i}$ und $\mathsf{d'_i}$ aus 3.2)

Die Herleitung für diesen Ansatz ist in [Nüc06] Seite 14 zu finden, Eggert hat in [ELF97] auch ihre Konvergenz und numerische Stabilität nachgewiesen.

3.4 Implementation des Algorithmus

Die Implementation basiert auf der in 2.3 kurz beschriebenen Softwarearchitektur des Robbie-Programmes. Kernstücke für die Implementation bilden hierbei das ICP-Modul sowie der ICP-Worker. Das ICP-Modul kümmert sich dabei um die Kommunikation mit den restlichen Modulen, also das Senden und Empfangen von Nachrichten. Der ICP-Worker verarbeitet dann die in den Nachrichten übertragenen Daten und berechnet die korrekten Scans. Im Folgenden werden nun diese beiden Programmteile sowie ihre Komponenten näher beschrieben.

3.4.1 ICP-Modul

Das ICP-Modul ist als „Messagemodule“ implementiert, arbeitet also nur passiv. Das bedeutet, dass es solange wartet, bis es eine für sich bestimmte Message empfängt und tritt erst dann in Aktion. Diese Nachrichten können entweder Kontrollnachrichten sein, ausgesendet von dem GUI, die es dem Benutzer ermöglichen, das ICP-Programm zu steuern, oder 3D-Scannachrichten sein. Die Kontrollnachrichten enthalten Steuercodes die an den ICP-Worker weitergegeben werden und ihn veranlassen, bestimmte Aktionen wie z.B. Speichern oder Laden von Scans auszuführen. Sie können aber auch den Modus des ICP-Moduls ändern. Dieser Modus legt fest, wie das ICP-Modul auf eingehende 3D-Scannachrichten reagieren soll. Die 3D-Scannachrichten enthalten Daten eines eben aufgenommenen 3D-Scans und wurden vom Laserscannermodul auf Grund einer Benutzereingabe an dem GUI erzeugt und abgeschickt. Empfängt nun das Modul einen solchen Scan, speichert es ihn zwischen und wartet zunächst auf eine „Pioneerdatamessage“, die aktuelle Odometriedaten enthält. Ist auch diese eingetroffen, leitet das Modul diese Daten, je nach Modus, an den ICP-Worker weiter. Folgende drei Modi existieren:

- Store
- Compute
- Store and Compute

Im Modus „Store“ werden die Scan- und Odometriedaten an den Worker weitergegeben und in einem Laserscancollectorfile (.lsc) gesammelt. Dieses Collectorfile funktioniert dabei ähnlich einer Playlist. Dabei erhalten die Scans eine laufende Nummer, abhängig von der momentan aufgenommenen Scanserie. Sie werden mit Hilfe des aktuellen Timestamps eindeutig benannt, wobei jeder Scan an sich einzeln unter diesem Namen abgespeichert wird und unter diesem Namen dann in dem Collectorfile seine Eintragung erhält, und zu jedem Scan werden hinter seinem Namen die dazu gehörenden Odometriedaten in der Reihenfolge x, y, z, Gier-, Roll- und Nickwinkel geschrieben. Der Gierwinkel ist dabei der Winkel zur Drehung um die Hochachse, Rollwinkel der zur Drehung um die Längsachse und Nickwinkel der zur Drehung um die Querachse. All diese Odometriedaten sind relativ zur absoluten Startposition des Roboters. Zu einem späteren Zeitpunkt, z.B. nach Beendigung aller Scanaufnahmen, kann dann im ICP-Modul das Collectorfile geöffnet werden und es kann nachträglich mit dem Zusammenfügen der Scans begonnen werden. Die genaue Spezifikation ist unter Anhang B zu finden.

Im Modus „Compute“ werden die eingehenden Daten an den Worker weitergeleitet und direkt von ihm zusammengefügt. Dieser Modus lässt es zu, direkt während einer Roboterfahrt schon korrigierte Odometriedaten aus dem ICP-Algorithmus zu bekommen, die dann von anderen Anwendungen verwendet werden können. Da die Korrektur der Odometriedaten mit Hilfe des ICP-Algorithmusses allerdings einige Zeit benötigt, ist dieser Modus bei zeitkritischen Anwendungen nicht zu empfehlen.

Im Modus „Store amd Compute“ geschieht dann beides gleichzeitig, wobei erst abgespeichert und dann zusammengefügt wird.

3.4.2 ICP-Worker

Der ICP-Worker ist das Arbeitstier innerhalb der ICP-Implementation. Er kümmert sich um die tatsächliche ICP-Berechnung, wobei er unter anderem die Punktpaarsuche, die Ermittlung der Korrekturdaten sowie die Kombination der Scans übernimmt. Zusätzlich liefert er auch noch die Funktionalitäten zum Öffnen und Speichern von l3d-Scandateien. Dies ist daher sinnvoll, da er zum Durchführen des ICP-Algorithmusses diese Funktionen sowieso beherrschen muss.

Bekommt er vom ICP-Modul eine Message mit dem Befehl zum Öffnen oder Speichern einer Datei, so wird dieser Befehl entsprechend der angegebenen Parameter, wie z.B. des Dateinamen, ausgeführt. Die daraus resultierenden Rückgabedaten, im Falle des Speicherns also ob dies erfolgreich war oder nicht, im Falle des Öffnens die entsprechenden Scandaten, werden an das Modul zurückgegeben, welches diese dann per Datenmessage an das GUI weiterleitet. Falls die Scandaten der Datei in Kugelkoordinaten gespeichert sind, wird auch die Umrechnung in karthesische Koordinaten vom ICP-Worker übernommen. Grundsätzlich basieren sämtliche weitere Verarbeitungsschritte der Scandaten auf karthesischen Koordinaten.

Falls der ICP-Worker den Befehl bekommt, einen Collectorfile zu öffnen, so startet er automatisch die Registrierung der darin enthaltenen 3D-Scans mit Hilfe des ICP-Algorithmus. Der grobe Ablauf des implementierten ICP-Algorithmusses ist in Bild 3.2 als Datenflussdiagramm dargestellt. Der gesamte Ablauf startet mit dem Öffnen eines Collectorfiles (lsc-Datei). Mit Hilfe dieses Files werden dann die entsprechenden lokalen Scans und ihre jeweiligen Odometriedaten geladen und an den Algorithmus weitergegeben. Im Großen und Ganzen besteht der gesamte Ablauf aus zwei Kreisläufen, einem inneren, der die Aktivitäten „korrespondierende Punktpaare ermitteln“ sowie „Korrekturberechnung“ enthält, und einem äußeren, der neben dem inneren Kreislauf zudem noch die Aktivitäten „kD-Tree laden“ und „Scans kombinieren“ enthält. Der innere Kreislauf wird für jede Iteration der ICP-Berechnung durchlaufen. Er bricht ab, sobald der Fehler beim Registrieren der beiden Scans unter einem vorher gewählten Grenzwert liegt. Der äußere Kreislauf wird für jeden weiter hinzukommenden lokalen Scan einmal durchlaufen, er bricht ab, sobald im Collectorfile keine weiteren Scans mehr eingetragen sind. Für den Fall das die neu eintreffenden Scans direkt eingefügt werden sollen, das ICP-Modul sich also im Modus „Compute“ befindet, wird am Ende des äußeren Kreislaufs, im Diagramm ist das die linke Verzweigung, so lange gewartet, bis ein weiterer Scan eintrifft.

Im Schnitt kann man sagen, dass für jeden weiteren lokalen Scan, also mit jedem Durchlauf des äußeren Kreislaufs, der innere Kreislauf 100 bis 200 mal durchlaufen wird. Um also Rechenzeit zu minimieren, ist die Geschwindigkeit der Aktivitäten des inneren Kreislaufes besonders wichtig. Vergleicht man den benötigten Aufwand der Aktivitäten „korrespondierende Punktpaare ermitteln“, bei einfachster Implementierung $\mathrm{O}(n^2)$, und „Kor-

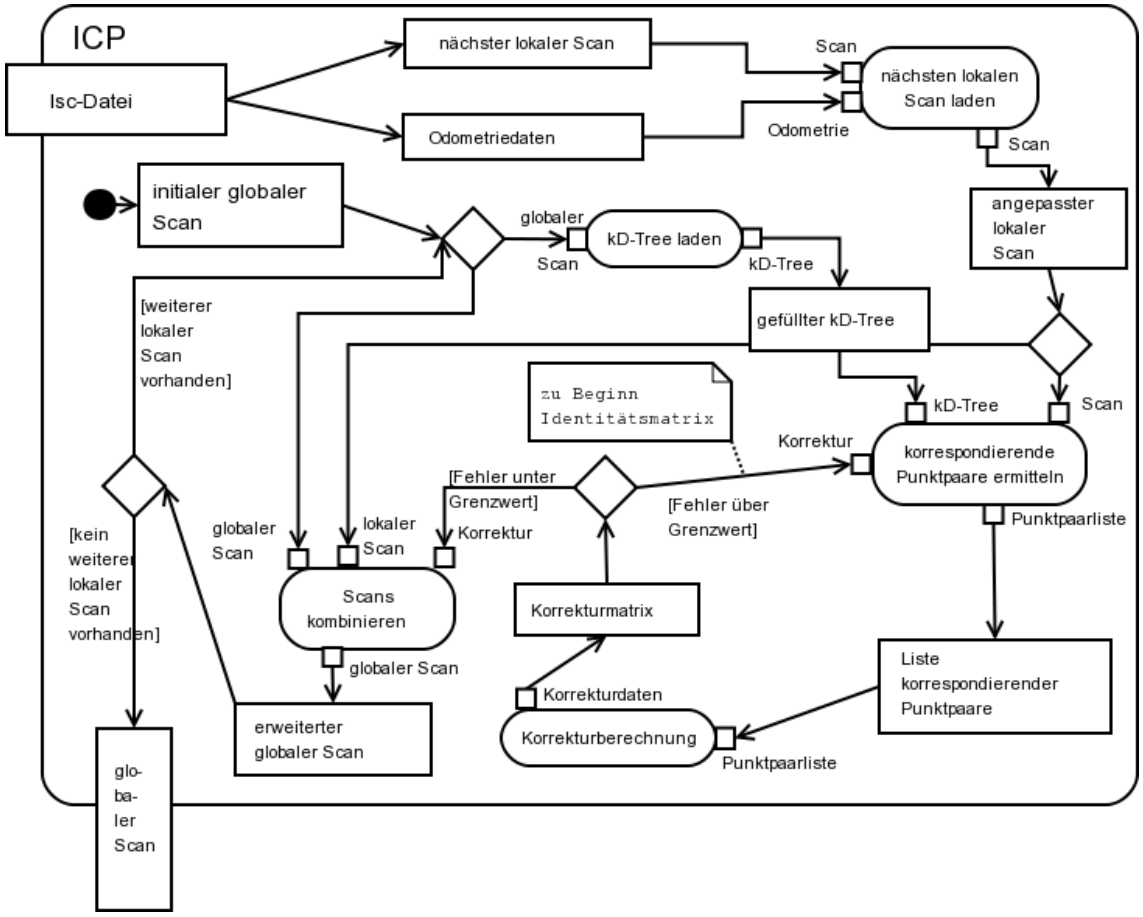

Bild 3.2: Datenflussdiagramm des ICP-Algorithmus

rekturberechnung“, da es eigentlich nur um eine Matrixbelegung sowie um eine Matrixzerlegung geht $\mathrm{O}(n+1)$, so wird klar, dass sich bei der Punktpaarermittlung am meisten Zeit einsparen lässt, unter anderem durch Punktreduktion, also einer möglichst kleinen Punktanzahl n, sowie durch Suchverfahren, die weniger Aufwand benötigen. Im folgenden Abschnitt gehe ich daher zunächst auf die Punktreduktion ein, darauf folgt dann die Beschleunigung der Paarsuche mittels kD-Tree.

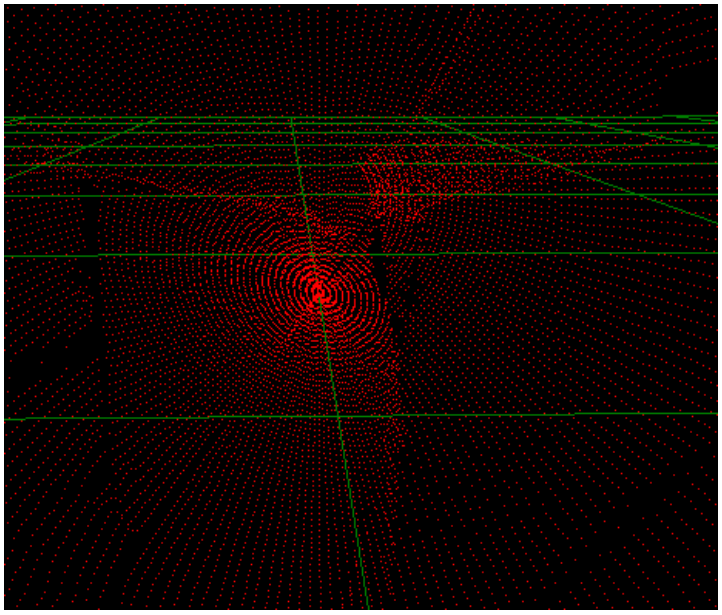

Bild 3.3: Bereich der Drehachse bei voller Punktauflösung

3.4.3 Punktreduktion

Die einfachste Methode den Aufwand $\mathrm{O}(n^2)$ gering zu halten ist, n möglichst klein zu halten. In unserem Fall bedeutet dies also mit möglichst geringer Anzahl Punkten zu arbeiten. Dabei darf die Punktauflösung allerdings nicht zu gering werden. Bei einem 3D-Scan mit Standartauflösung, also einer Azimutwinkelauflösung von 100, erhalten wir maximal pro Scan 68200 Punkte. Auf Grund der Art der Aufnahme haben wir eine Punktballung im Bereich der Drehachse des Scanners (siehe Bild 3.3). Außerdem sind auf diesem Bild sehr gut die einzelnen 2d-Scans zu erkennen, aus denen sich der 3D-Scan zusammensetzt. Die Punkte eines 2d-Scans bilden nämlich immer eine der annähernd geraden gepunkteten Linien. Um nun die Punktanzahl zu minimieren, kann man zum Einen die Dichte der einzelnen 2d-Scans zueinander betrachten, zum Anderen die Dichte der Punkte innerhalb eines dieser 2d-Scans betrachten. Letztendlich betrachte ich mit Einschränkungen beide Aspekte. Vor der tatsächlichen Beschreibung der Punktreduktion soll angemerkt werden, dass diese Punktreduktion auf der Interpretation des Kugelkoordinatesystems basiert. Ei-

ne spätere Punktreduktion, also eine Punktminimierung in Scans, die bereits in karthesischen Koordinaten vorliegen, ist nicht mehr möglich. Da alle hier verwendeten 3D-Scans ursprünglich in Kugelkoordinaten vom Scanner kommen und bei der Umrechnung in karthesische Koordinaten ohnehin jeder Punkt betrachtet werden muss, scheint mir an der Stelle der Koordinatentransformation eine Punktreduktion am sinnvollsten.

Kommen wir jetzt zum tatsächlichen Reduktionsverfahren. Um die Punktballung im Bereich der Drehachse zu minimieren, ist eine Betrachtung der Punktdichte zwischen einzelnen 2d-Scans nötig. Dies würde allerdings innerhalb des Umrechnungsverfahrens zu einigen Sprüngen führen. Um dies zu vermeiden und trotzdem eine sinnvolle Minimierung im Drehachsenbereich zu bekommen, wird in diesem Bereich die Anzahl der dort einflussnehmenden 2d-Scans reduziert. So werden in einem Polarwinkelbereich von $\pm 40^\circ$ um die Drehachse herum nur die Punkte beachtet, die auf einem 2d-Scan mit gerader Nummer liegen.

Innerhalb eines 2d-Scans betrachte ich nun, abhängig vom Abstand zum Ursprung, die Änderung dieses Abstandes, die der folgende Punkt aufweist. Liegt diese unter einem Grenzwert, so wird der nächste Punkt übersprungen. Dieses Überspringen kann in Abhängigkeit vom Abstand zum Ursprung bis zu fünf Punkte betragen. Die Betrachtung der Punkte in Abhängigkeit ihres Abstandes zum Ursprung des Kugelkoordinatensystems ist daher nötig, da bei fester Winkelauflösung und hohem Abstand die Punktauflösung abnimmt und damit weniger Punkte wegfallen dürfen.

Mit diesem Verfahren erreiche ich bei Reduzierung der Punktanzahl auf, im Schnitt, $\frac{1}{4}$ eine möglichst hohe Detailgenauigkeit der Auflösung (siehe Bild 3.4).

3.4.4 Beschleunigung der Paarsuche mittels kD-Tree

Wie zum Ende von Kapitel 3.4.2 bereits erwähnt, lässt sich der gesamte Ablauf am leichtesten beschleunigen, indem man die Punktpaarsuche beschleunigt. Bei dieser Suche geht es grundsätzlich darum, zu jedem Punkt des lokalen Scans einen nächstgelegenen Punkt aus dem globalen Scan zu finden, wie beispielsweise in Bild 3.5 zu sehen. Dies geschieht unter der Voraussetzung, dass man die beiden Scans mit Hilfe der initialen Verschiebungsdaten provisorisch zusammengefügt hat, und unter der Annahme, dass der, zu einem Punkt

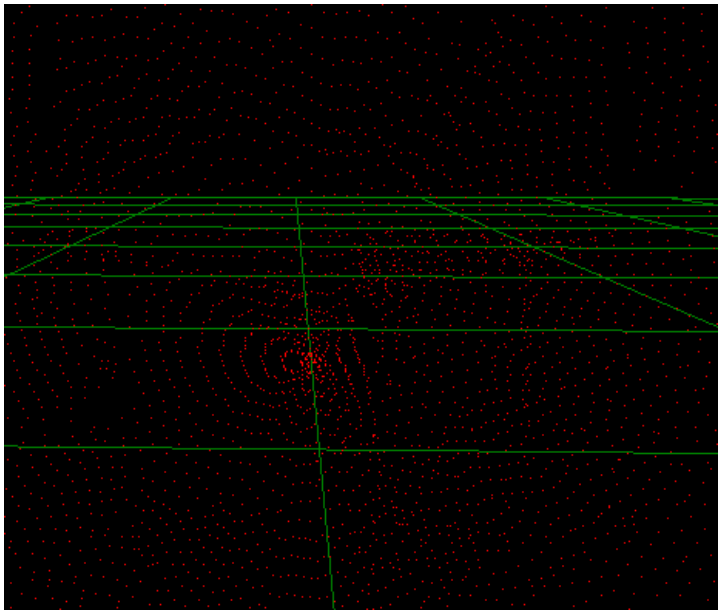

Bild 3.4: Gleicher Ausschnitt wie Bild 3.3 jedoch minimierte Punkteanzahl

des einen Scans, nächstgelegene Punkt des anderen Scans eigentlich den selben Punkt darstellt. Diese Punktpaare werden also in so fern korrigiert, als dass der Abstand zwischen ihnen minimiert werden soll.

Da es nun aber vorkommen kann, dass zwei Scans gar nicht vollständig überlappen, es also unter der zuvor gemachten Annahme gar keinen Partnerpunkt aus dem globalen Scan für jeden Punkt des lokalen Scans gibt, muss man dies berücksichtigen, in dem man nur innerhalb einer gewissen Umgebung nach Punkten des globalen Scans sucht. Dies beschleunigt zum Einen schon mal ein wenig unsere Paarsuche, da die Suche, hat man innerhalb einer gewissen Umgebung keinen Partnerpunkt gefunden, einfach abgebrochen werden kann, zum Anderen werden falsche Punktkorrespondenzen vermieden, die unsere Korrekturberechnung verfälschen könnten. Deswegen wird in Bild 3.5 auch der komplette globale Scan (rot) angezeigt, aus dem lokalen Scan (weiß) werden allerdings nur diejenigen Punkte angezeigt, die einen Partner gefunden haben.

Um den Vorgang der Partnersuche weiter zu beschleunigen, schauen wir uns nun den ei-

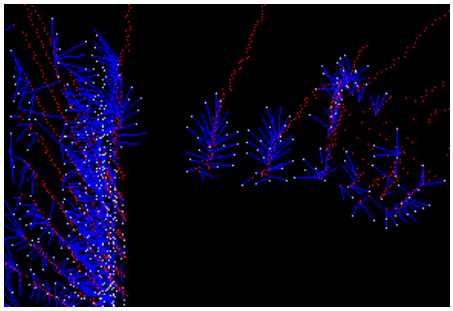

Bild 3.5: Beispiel von Punktkorrespondenzen, rot = globaler Scan, weiß = lokaler Scan, blaue Linien verbinden Punktpaare

gentlichen Suchvorgang an. Die simpelste Methode wäre es, über jeden Punkt des lokalen Scans zu gehen, den Abstand zu jedem Punkt des globalen Scans zu berechnen und dann den Punkt aus dem globalen Scan als Partner zu bestimmen, der den geringsten Abstand aufweist. Dieses Verfahren hätte aber einen enormen Aufwand von $\mathrm{O}(n^2)$ und würde, bei einer zu erwartenden Punkteanzahl von mehreren Hunderttausend, enorm viel Zeit beanspruchen. Sehr viel eleganter und schneller sucht man mit Hilfe eines kD-Trees. Hierbei werden alle Punkte eines Scans in den kD-Tree geladen und dann mit Hilfe dessen zu jedem Punkt des anderen Scans ein Partner gesucht. Beim Erzeugen eines kD-Trees werden die darin enthaltenen Punkte, bei Überschreitung einer Maximalanzahl, in einer ihrer k Dimensionen, in unserem Fall drei Dimensionen, aufgeteilt. Danach werden diese aufgeteilten Punktmengen neu betrachtet und, falls ihre Anzahl immer noch größer als die Maximalanzahl ist, erneut aufgeteilt. Dies geschieht rekursiv, bis alle Punkte entsprechend aufgeteilt sind. Will man nun zu einem Punkt einen Partnerpunkt innerhalb des kD-Trees finden, so muss man nicht mehr über alle Punkte gehen, sondern nur noch die Punkte des Astes durchsuchen, in dessen Bereich unser suchender Punkt fällt. Falls es innerhalb der Maximalumgebung, in der die Suche für den Punkt stattfinden soll, gar keine Äste des kD-Trees gibt, kann die Suche sofort abgebrochen werden, da kein Partnerpunkt nahe genug wäre. Um diese Betrachtung bei der Suche zu ermöglichen, wird beim Generieren des kD-Trees in jedem Knoten eises Astes zusätzlich noch der von ihm abgedeckte Bereich gespeichert.

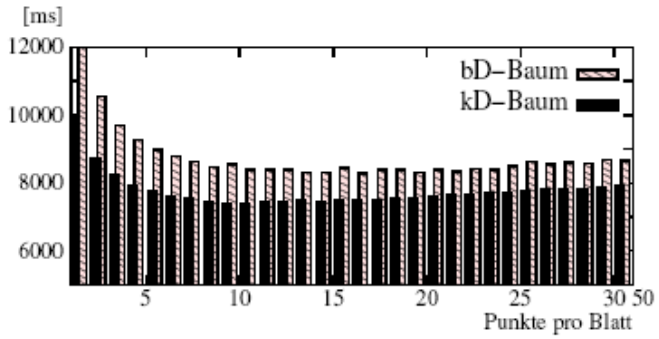

Bild 3.6: Vergleich der Rechenzeit bei Suche in kD-Tree und BD-Tree (Quelle: [Nüc06] Seite 35)

Der Suchaufwand sinkt mit diesem Verfahren auf $\mathrm{O}(n \log n)$. Einziger Nachteil dabei ist die Zeit, die man für das Generieren des kD-Trees benötigt. Hierbei wäre es praktisch, jedes Mal den lokalen Scan zu verwenden, da dieser immer eine geringe Punktanzahl aufweist. Beachtet man aber die Häufigkeit, mit der die Generierung vorkommt (ein Mal pro weiterem lokalen Scan) und vergleicht diese mit der Häufigkeit der Partnersuche (bei jeder Iteration), bei der dann jeweils die Suche für jeden der evtl. hunderttausend Punkte des globalen Scans zumindest gestartet werden müsste, auch wenn sie auf Grund der maximalen Suchumgebung sofort abgebrochen werden würde, so wird einem schnell klar, dass das Laden des globalen Scans in den kD-Tree die schnellere Variante ist.

Ein weiteres Verfahren zur Beschleunigung der Suche ist unter anderem auch das Verwenden eines balanzierten Box-Dekompositions-Baumes (BD-Baum). Dieses Verfahren wurde von Nüchter in [Nüc06] getestet und mit dem kD-Tree verglichen, wobei er zu dem Ergebnis kam, dass der kD-Tree in allen Fällen eine schnellere Suche gewährleistet (siehe 3.6). Für meine Implementation hab ich mich daher für die Verwendung eines kD-Trees zu Beschleunigung der Partnersuche entschlossen.

3.4.5 Berechnung der Transformationskorrektur

Zur Berechnung der Korrektur wird versucht, den Abstand zwischen den Punktpaaren zu minimieren. Es werden also nur die gefundenen Punktpaare benötigt. Daher werden der Funktion auch nur die Listen, die die gefundenen Punktpaare enthalten übergeben. Wir haben also zwei Mengen an Punkten aus beiden Scans. Für diese beiden Punktewolken wird nun zunächst der Schwer- beziehungsweise Mittelpunkt ausgerechnet. Als nächstes werden die für das gewählte mathematische Lösungsverfahren benötigten Matrizen belegt, für das SVD-Verfahren wäre das die $\boldsymbol{H}$-Matrix. Danach erfolg die Berechnung, wobei das Lösen der Singulärwertzerlegung vom Lapackpp-Packet übernommen wird. Hier erhält man dann aus $\boldsymbol{R} = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{V}$ die Rotationsmatrix. Zu guter letzt erfolgt nun die Berechnung der Translation. Hierfür rotiert man das Zentrum der Punktpaarwolke des lokalen Scans und vergleicht dieses dann mit dem Zentrum der anderen Punktpaarwolke. Der Positionsunterschied dieser beiden Zentren ist die gesuchte Translation. Mit den daraus ermittelten Daten wird dann die 4×4-Transformationsmatrix belegt die dann als Ergebnis zurückgegeben wird. Diese Transformationsmatrix wird dann auf die aus initialen Daten oder dem letzten Iterationsschritt bereits vorhandene Transformationsmatrix hinzumultipliziert.

3.4.6 Wann Iteration abbrechen?

Um aus dem Iterationskreislauf herauszukommen ist es nötig, einen Grenzwert festzulegen, ab dem wir das zusammengefügte Ergebnis als ausreichend korrekt ansehen. Der Fehlerwert für das Zusammenfügen der Scans errechnet sich aus der Summe der quadratischen Distanzen der korrespondierenden Punkte, er wird bei jeder Punktpaarermittlung errechnet. Dabei stossen wir auf mehrere Probleme. Als erstes muss beachtet werden, dass dieser Fehler, fügt man nicht die selben Scans zusammen, im Prinzip nie Null werden kann, da die Rasterisierung der Punkte in unterschiedlichen Scans verschieden ist. Da das Punktraster innerhalb eines Scans, der weiter entfernte Objekte enthält, auch größer ist, Grund dafür ist das Kugelkoordinatensystem mit fester Winkelauflösung, ist der Fehlerwert bei korrektem Zusammenfügen zweier solcher Scans ebenfalls größer als bei Scans, die nur nahe Objekte enthalten. Gut zu sehen ist die in Bild 3.7. Da sich der Fehlerwert aus den Punktpaaren errechnet bekommt man auch einen geringeren Fehlerwert wenn man

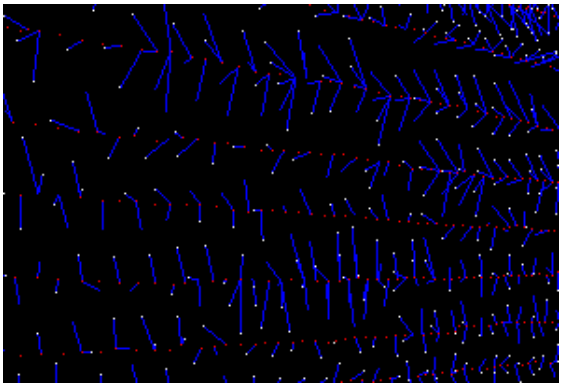

Bild 3.7: Trotz guter Registrierung ergeben sich aus den hier gefundenen Punktpaarungen erhöhte Fehlerwerte

weniger Punktpaare findet. Aus all diesen Gründen muss der Fehlergrenzwert, bei dem abgebrochen werden soll, dynamisch und in Abhängigkeit der Anzahl Punktpaare, die zu seiner Ermittlung geführt haben, gewählt werden und kann nicht vorher festgelegt sein.

Um all diese Aspekte zu beachten wird bei der Implementation nicht auf einen festen Fehlerwert geachtet sondern es wird der Quotient aus der Anzahl korrespondierender Punkte und Fehlerwert des aktuellen Iterationsschrittes mit dem Quotienten des letzten Iterationsschrittes verglichen. Ist der Quotient des aktuellen Schrittes kleiner als der des letzten Schrittes bedeutet dies, dass entweder der Fehler zugenommen hat, die Anzahl korespondierender Punkte abgenommen hat oder sogar beide diese Fälle eingetreten sind. Sobald dies eintritt werden eine Anzahl weitere Testiterationen durchgeführt, um zu verhindern, das es sich nicht um ein lokales Phänomen handelt. Wenn innerhalb dieser Testiterationen auch alle dort ermittelten Quotienten kleiner sind als der Quotient bei Beginn der Testläufe, so wird die Korrekturberechnung abgebrochen mit der Annahme, die beste Transformation gefunden zu haben.

Kapitel 4

Experimente und Ergebnisse

In diesem Kapitel soll nun näher auf die Leistung dieser ICP-Implementation eingegangen werden. Anhand verschiedenster Scantypen, komplett syntetischen, simulierten als auch realen, wird die Grenzen der Korrigierbarkeit mit Hilfe des ICP-Algorithmusses als auch die Probleme bei der Belegung der Einstellungsvariablen gezeigt.

4.1 Versuchsaufbau

Zum Testen werden auf drei verschiede Arten erzeugte Testdaten verwendet. Der syntetisch erzeugte Scan beruht auf einer errechneten dreidimensionalen Struktur, die im 3D-Laserscanformat abgespeichert wurde. Es handelt sich dabei um zwei übereinander stehende Quadrate. Insgesamt besteht dieser künstlich erzeugte Scan aus 8000 Punkten. Diese liegen allerdings so eng zusammen, dass in Bild 4.1 die einzelnen Punkte nicht mehr als solche zu erkennen sind, sondern nur noch als eine Linie.

Die in einer Simulation erzeugten Scans basieren auf der Verknüpfung der Robbie-Software mit der USARSim Umgebung. Diese Möglichkeit ist aus der Studienarbeit von Denis Holzhäuser entstanden. Hierbei fährt ein simulierter Roboter durch eine dreidimensionale simulierte Landschaft und liefert entsprechend simulierte Sensordaten zurück. Ein Beispiel für solche Scandaten ist in Bild 4.2 zu sehen.

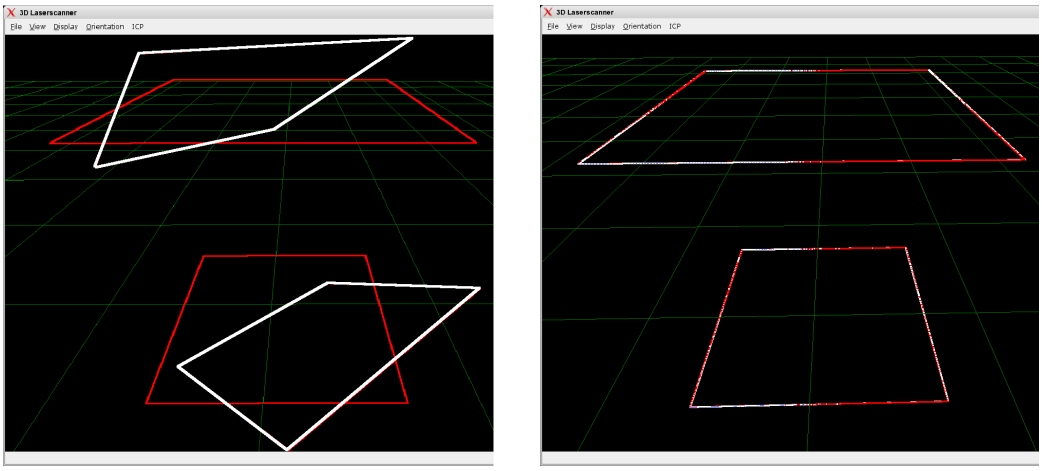

Bild 4.1: Synthetische Scans, links vor der Korrektur, rechts nach Abschluss der Korrektur, wobei: rot = „globaler“ Scan, weiß = „lokaler“ Scan

Die realen Daten sind schließlich auf Basis der Installation des 3D-Scanners auf Robbie (siehe Bild 2.2) entstanden. Hierfür wurde der Roboter über den Flur der Uni gefahren. Ausserdem wurde während der German Open des RoboCups die gesamte „Rescue-League“-Arena aufgenommen. Bei dieser Arena traten dann tatsächlich reale Verschiebungen in der Z-Achse sowie im Nick- und Roll-Winkel auf.

4.2 Versuchsdurchführung

Bei den Experimenten mit den synthetischen Scans wurden zum aufeinander Registrieren ein und der selbe Scan verwendet und einer von ihnen dann lediglich um seine drei Achsen gedreht. Da die Rotationskorrektur der eigentliche Knackpunkt der Korrekturberechnung ist, dies wurde bei den Versuchen auch bestätigt, wurde hier hauptsächlich Verdrehungen der Scans simuliert. Als initiale Transformationswerte des zu registriedenen Scans wurde hierbei die entsprechend zu testende Verschiebung beziehungsweise Rotation eingegeben. Die Manipulation betraf dabei alle sechs Dimensionen, sowohl einzeln als auch in Kombination miteinander. Ein Beispiel hierfür ist Bild 4.1, hier wurde um 40° um die Hochachse gedreht und um 30° um die Querachse. Die Ergebnisse dieser Versuche sind in

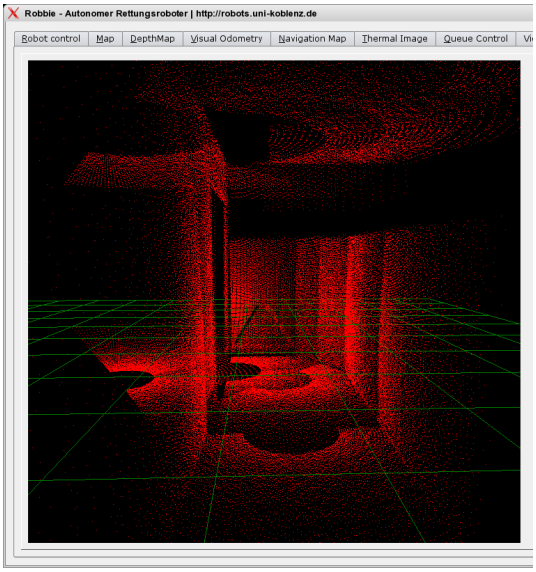

Bild 4.2: Simulierter Scan aus USARSim, nach dem Registrieren

Kapitel 4.3.1 zu sehen.

Für die Versuche mit Scandaten aus der Simulationsumgebung wurden zwei unterschiedliche Scanserien verwendet, eine mit optimalen Daten und eine, in deren Scans ein künstliches Rauschen hineingerechnet wurde. Per Hand wurden dann die zu den jeweiligen Scans gehörenden Odometriedaten im Collector-File geändert um falsche Odometriedaten zu simulieren. Diese Collector-Files wurden dann geladen und mit Hilfe dieser dann die Registrierung durchgeführt. Da auch hier die Korrektur der Rotation das Hauptproblem darstellte, wurde bei den Experimenten dort der Schwerpunkt gesetzt. So wurde dann bei einzelnen Scans im Collector-File deren jeweilige Verdrehung schrittweise geändert um die fehlerhaften Odometriedaten zu generieren.

Um eine nachvollziebare Versuchsreihe mit realen Scans zu erstellen, wurde ein Bereich des Flures des B-Gebäudes der Universität Koblenz aufgenommen und die Position und Ausrichtung des Roboters dabei jeweils bevor ein Scan geschossen wurde per Hand geändert. Dieses Verschieben geschah in steigenden Gradschritten und mit unterschiedlich großen Verschiebungsschritten. Um die Rechenzeit beim Registrieren nicht zu lang wer-

den zu lassen wurde bei dieser Versuchsserie eine nur fünf Scans enthaltenden Scanserie verwendet. Die Ergebnisse sind unter Kapitel 4.3.3 zu sehen. Zudem wurde versucht, die 19 lokalen Scans die bei der Aufnahme der „RoboCup-Rescue“-Arena entstanden zu einem globalen Scan zu matchen. Die hierbei durch Schrägen und Hindernisse entstandenen Verschiebungen und Verdrehungen sind im Einzelnen nicht mehr nachvollziehbar, allerdings sind die Änderungen der verwendeten Parameter des ICP-Algorithmusses nachvollziehbar. Hierbei wurde sowohl mit Suchumgebungsgrößen von 20000, 30000, 40000 und 100000mm^2 als auch mit 1000, 2000 und 4000 minimalen Punktpaaren experimentiert. Dies ist bei dieser Arena besonders interessant und hat einen Ausschlag, da sie extrem verschachtelt ist. Daher werden hier bei zu groß gewählten Suchumgebungen sehr leicht falsche Punktkorrespondenzen gebildet oder es sind nicht genügend Punktpaare im Überlappungsbereich vorhanden, aus denen sinnvolle Punktkorrespondenzen erzeugt werden können. Dadurch ergeben sich Fehler in der Korrekturberechnung.

4.3 Ergebnisse

4.3.1 Synthetische Ergebnisse

Die Ergebnisse bei den Versuchen mit synthetischen Scans waren recht eindeutig. Wie erwartet hatten die Translationsfehler keinen Einfluss auf das Ergebnis und wurden immer perfekt korrigiert. Sie sind einzig von der Suchumgebung, innerhalb derer man die Paarungen der Punkte sucht, abhängig. Wählt man diese groß genug, so wird jeder noch so große Translationsfehler korrigiert. Aber auch die Rotationsfehler wurden bis hin zu einem Winkel von 45° nahezu perfekt korrigiert, wobei auch alle drei Achsen gleichzeitig um jeweils 45° verdreht werden konnten. Das Ergebnis eines solchen Versuchs ist in Bild 4.1 zu sehen. Ab einer Verdrehung von über 45° wurde der Scan dann in die falsche Richtung korrigiert, die Scans lagen zwar wieder perfekt aufeinander, jedoch war der zu registrierende Scan einfach um 90° gedreht worden. Der quadratische Fehlerwert zwischen den einzelnen Scans lag am Ende des Algorithmus dabei immer im Nachkommabereich, sowohl bei korrekter als auch bei fehlerhafter Rotation. Dies resultiert daher, da es sich ursprünglich um identische Scans handelt, die lediglich gegeneinander verdreht wurden,

Verdrehung (in °)	Suchumgebung	Iterationen	Ergebnis
0	15000	74	OK
7,5	15000	106	OK
18,5	15000	179	OK
30,4	15000	7	NOK
30,4	20000	164	OK
41,8	20000	210	OK
47,6	20000	2	NOK
-9,7	15000	250	OK
-15.5	15000	32	NOK
-15.5	20000	17	NOK

Tabelle 4.1: Ergebnisse des Matching zweier USARSim-Scans bei unterschiedlichster fehlerhafter Verdrehung und geänderter Suchumgebung

also tatsächlich zu jedem Punkt des einen ein identischer Punkt des anderen zu finden ist, die exakt aufeinander passen. Daher ist in Bild 4.1 auch fast keine Unterscheidung der verschobenen Scans mehr möglich. Dies ist bei simulierten als auch bei realen Scans nicht gegeben.

4.3.2 Simulierte Ergebnisse

Zu allererst bleibt festzustellen, dass sowohl für die verrauschten als auch für die optimalen Ursprungsdaten die gleichen Ergebnisse herauskamen. Der Algorithmus ist also recht unempfindlich gegenüber verrauschten Daten. Daher wurde sich bei der folgenden Ergebnisbeschreibung nur noch auf die Versuche mit verrauschten Daten beschränkt. Die Tabelle 4.1 zeigt den Versuch mit synthetischen Daten, die verrauscht sind. Hierbei wurde Schrittweise die initiale Verdrehung der Scans geändert, angegeben in der ersten Spalte. Die Suchumgebung beschreibt den quadratischen Abstand, innerhalb dessen für einen Punkt ein Partner gesucht werden soll. Die Spalte für die benötigten Iterationen beschreibt, wie viele Iterationen bis zum Abbruch des Algorithmus zum Registrieren eines lokalen

Scans benötigt wurden. Abschliessend ist das Ergebnis eines jeden Versuchschrittes zu sehen. Sehr auffallend ist bei dieser Versuchsreihe, dass die Korrektur in Verdrehungen im Uhrzeigersinn noch bis über 40° korrigiert werden können, bei Verdrehungen gegen den Uhrzeigersinn dies aber nur im Bereich bis zu 10° funktioniert. Der Grund hierfür ist in Bild 4.3 zu sehen. Um den Fehler besser zu erkennen, wurden die Höhenwerte der Punkte bei der Anzeige nicht beachtet. Der Fehler wird anhand der sichtbaren Doppelwände deutlich. Es handelt sich hierbei um einen lokalen Scan der innerhalb einer abgerundeten Biegung aufgenommen wurde. Da diese Biegung wenig Aussage über Rotationsfehler gibt und die innerhalb der Biegung liegenden Punkte schon nahe liegende Punktpartner besitzen, reicht die „Kraft“ der weiter entfernt liegenden Punkte nicht mehr aus, um die Rotation richtig zu korrigieren. Dies kann bei Scans mit wenigen Eckpunkten vorkommen. Was hier allerdings auch zu sehen ist, ist, dass der in der Scanserie folgende Scan wieder korrekt registriert wurde.

Bei der Verdrehung im Uhrzeigersinn kommt es zu keiner solchen Punktanhäufung innerhalb einer Biegung und somit ist auch wieder eine Korrektur bis zu einer Verdrehung von annähernd 45° möglich.

4.3.3 Gemessene Ergebnisse

Wie bei den beiden vorangegangenen Versuchsreihen, so ist auch hier das Hauptproblem die Rotation. Die Korrektur des Translationsfehler ist wiederum nur abhängig von der Größe der Suchumgebung innerhalb derer die Punktpaarungen zustande kommen. Auf die Wahl dieses Parameters wird später noch eingegangen. Die Ergebnisse der im Flur durchgeführten Experimente sind in Tabelle 4.2 zu sehen.

Die zu dieser Tabelle gehörenden Scans sind in Bild 4.4 zu sehen. Zusätzlich zu den schon bei Tabelle 4.1 verwendeten Daten sind hier noch die Fehler vor und nach dem Registrieren als „Sum-of-Square-Differences“ angegeben. Gut zu sehen ist hier das der Fehlerwert auch bei guter Registrierung aus den in Kapitel 3.4.6 beschriebenen Gründen nicht Null werden kann. Sehr bemerkenswert im Vergleich zu den simulierten Daten ist hier, dass eine Korrektur von Verdrehungen bis knapp über 50° möglich ist, wählt man die Parameter entsprechend. Grund dafür ist das im Vergleich zur simulierten Umgebung sehr viel

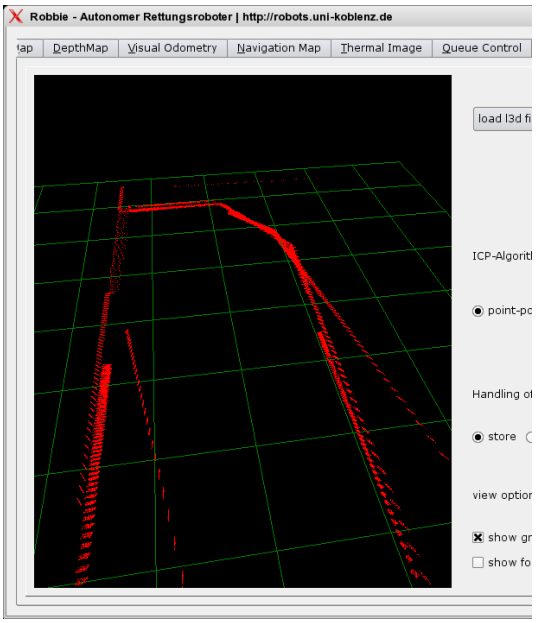

Bild 4.3: Auftretender Fehler bei USARSim Scans, 3D-Punktewolke ohne Höhenwerte angezeigt

verwinkeltere Terrain, das gescannt wurde. Somit sind in den Scans auch viel unterschiedlichere Flächen enthalten, deren Punkte zum korrekten Registrieren geeigneter sind.

Wieso nun also nicht eine riesige Suchumgebung, eventuell sogar eine unendlich große, wählen? Die Antwort darauf findet sich in den Experimenten mit der Scanserie der „RoboCup“-Arena. Zum Einen wird dauert der gesamte Registrierungsprozess sehr viel länger, bei einem Abstand von 100000mm^2 um $\frac{1}{3}$ länger als bei 30000mm^2, zum Anderen werden unpassende Punktpaare gebildet, die in die Korrekturberechnung einfließen und dieses verfälschen. Als beste Kombination dieser beiden Parameter hat sich eine Suchumgebung von 30000mm^2 bei einer minimalen Punktanzahl von 2000 herausgestellt. Kleiner sollte diese Punkteanzahl auf keinen Fall sein, da sonst nicht genügend aussagekräftige Punktpaare in die Korrekturberechnung einfließen. Ein mit diesen Parametern erstellter Scan ist in Bild 4.5 zu sehen. Das Registrieren dieses Scans dauerte etwa fünf Minuten.

Verdrehung (in °)	Suchumgebung	Iterationen	Startfehler	Endfehler	Ergebnis
0	20000	53	53	51	OK
17	20000	39	71	52	OK
28	20000	93	69	50	OK
45	20000	198	68	50	OK
48	20000	15	68	63	NOK
48	25000	320	73	53	OK
51	25000	106	72	67	NOK
51	30000	137	80	74	NOK
-17	20000	127	73	51	OK
-28	20000	113	77	51	OK
-45	20000	210	78	51	OK
-48	20000	155	78	71	NOK
-48	25000	370	84	66	NOK
-48	30000	502	91	59	OK
-51	30000	500	91	59	OK
-54	30000	108	91	79	NOK

Tabelle 4.2: Ergebnisse des Matching zweier gemessener Scans bei unterschiedlichster fehlerhafter Verdrehung und geänderter Suchumgebung. Zusätzlich eingetragen sind der Durchschnittsfehler bei Start und Abbruch des Algorithmus.

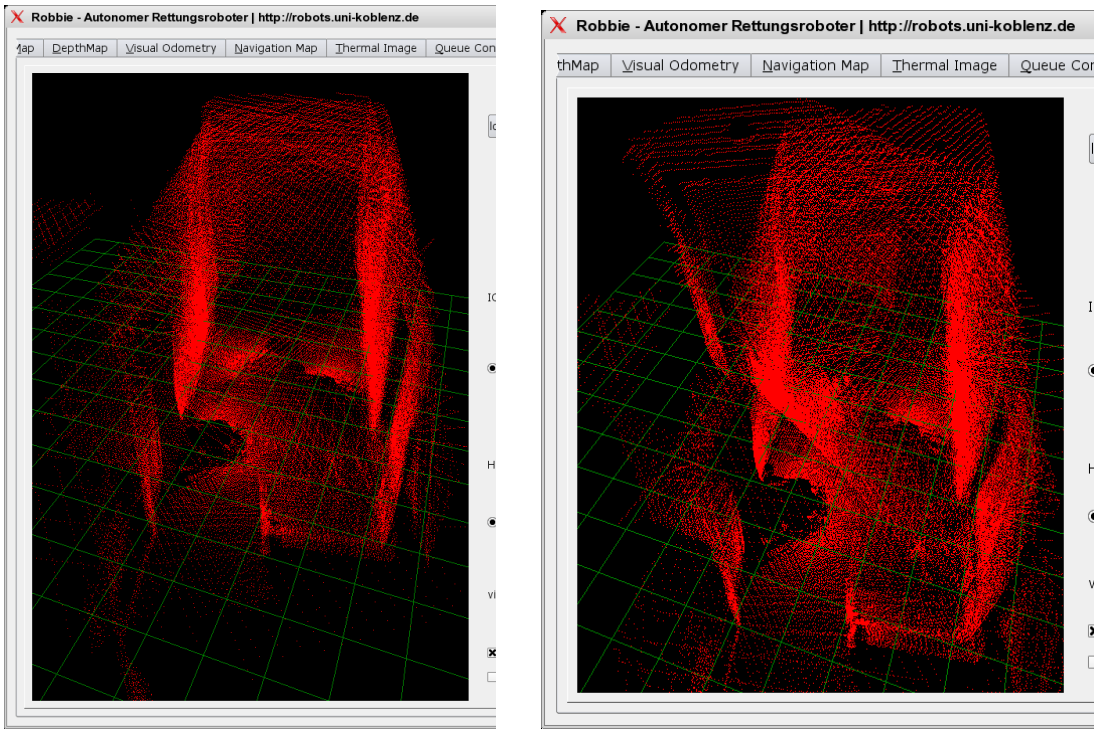

Bild 4.4: Registrierte Scans aus Versuchsreihe mit realen Scans, links korrekt, rechts fehlerhaft

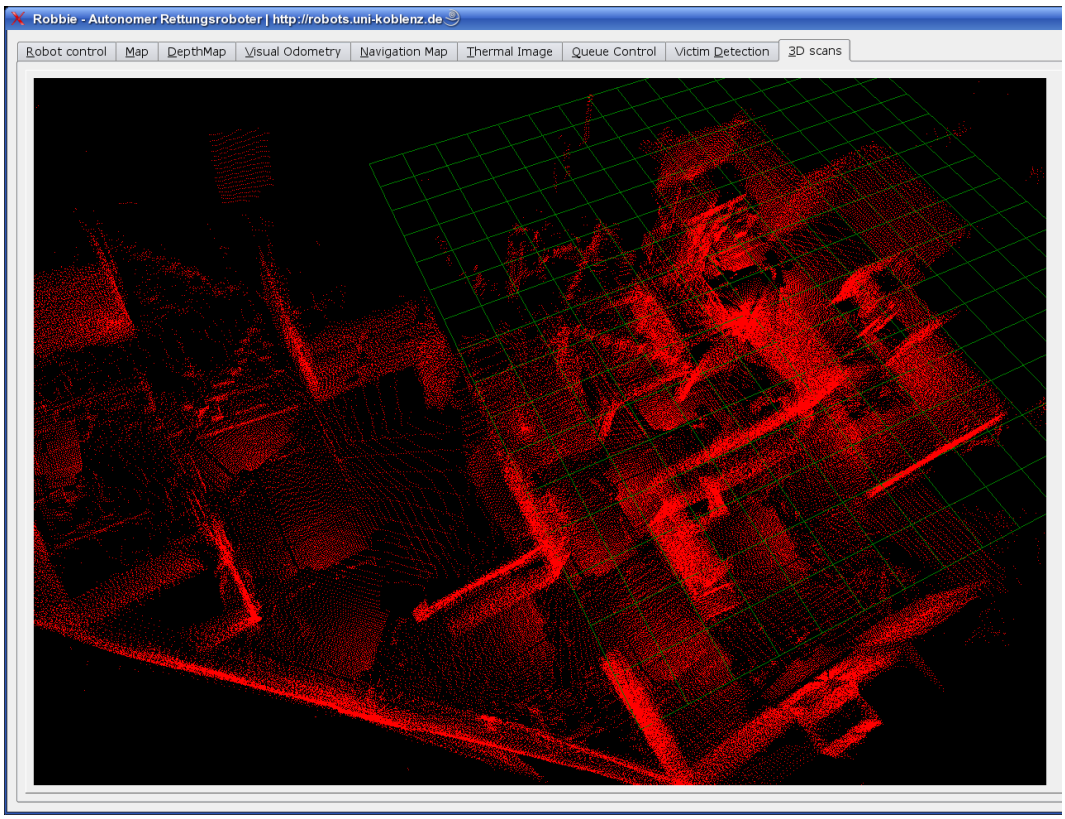

Bild 4.5: Die „RoboCup-Rescue“-Arena, bestehend aus 19 lokalen Scans

Kapitel 5

Zusammenfassung

Was erhalten wir nun durch dieses Verfahren? Der ICP-Algorithmus liefert uns die Möglichkeit, theoretisch beliebig große 3D-Scans aus beliebig kleinen lokalen 3D-Scans zu erstellen, solange wir für jeden dieser lokalen Scans eine initiale Bewegungsschätzung besitzen. Dies funktioniert relativ zuverlässig, solange die Fehler der zu den Scans gehörenden Odometriedaten nicht zu groß werden. Vor allem der Rotationsfehler ist, wie in Kapitel 4.3.3 zu sehen war, ausschlaggebend. Er sollte, abhängig von den eingestellten Parametern, möglichst nicht mehr als 45° pro Achse von einem Scan zum nächsten betragen. Ist der auftretende Rotationsfehler zu vernachlässigen, so ist die Korrektur des Translationsfehlers nur abhängig von der eingestellten Suchumgebung und kann somit beliebig groß sein.

Ein weiteres Problem dieses Algorithmus ist die mit ihm verbundene Rechenzeit. Da selbst bei zu registrierenden Scans, deren Odometriedaten nur minimale Fehler aufweisen, ein sofortiges Konvergieren nicht gewährleistet ist (siehe Tabelle 4.2), bei stark fehlerbehafteten Odometriewerten die Anzahl Iterationen schnell die 500er Marke überschreiten kann, ist eine Vorhersage der benötigten Rechenzeit auf keinen Fall möglich. Ein Einsatz in zeitkritischen Anwendungsgebieten ist somit in der momentanen Version bei aktueller Rechenleistung nicht zu empfehlen, da möglicherweise sehr lange Wartezeiten entstehen würden, während man auf Ergebnisse des ICP-Algorithmus wartet.

Desweiteren bleibt noch zu erwähnen, dass selbst bei Rotationsfehlern, die kleiner als die

maximal möglichen 45° sind, ein korrektes Registrieren der Scans nicht immer garantiert werden kann (siehe Tabelle 4.1). Ursache hierfür ist eine ungünstige Punktverteilung in den einzelnen lokalen Scans. Auffallend in dieser Tabelle ist, dass der Algorithmus bei fehlerhafter Registrierung schon nach wenigen Iterationen abbricht. Diese Tendenz ist auch bei der Registrierung realer Scans in Tabelle 4.2 zu sehen. Hier ist außerdem die Betrachtung des Start- und Endfehlers interessant. Da der Endfehler bei realen Scans nie Null werden kann, muss die Bewertung der abgeschlossenen Registrierung auf andere Weise geschehen. Eine Möglichkeit wäre hier die Betrachtung des Verhältnisses zwischen Start- und Endfehler. Ist dieses Verhältnis zu klein könnte man von einer fehlerhaften Registrierung ausgehen. Dies würde allerdings bei geringer Fehlerbehaftung der Odometriedaten zu Schwierigkeiten führen, da hier ebenfalls das Verhältnis zwischen Start- und Endwert klein ist. Um auch dort eine korrekte Bewertung zu ermöglichen, wäre eine vorher durchgeführte Testregistrierung bei nicht vorhandenem Translationsfehler ein Ansatz. Da bei den realen Scanergebnissen auffällt, dass alle korrekt durchgeführten Registrierungen ähnliche Endfehler liefern, könnte man den Endfehlerwert der Testregistrierung also als eine Art Richtwert verwenden.

Ist ein Fehler in der Registrierung eines lokalen Scans aufgetreten, heißt dies nicht grundsätzlich, dass der gesamte globale Scan untauglich geworden ist. Meißt traten solche Registrierungsprobleme nur bei einzelnen lokalen Scans auf, die Nachfolgenden wurden dann wieder korrekt eingefügt (siehe Bild 4.3). Um einen möglichst komplett fehlerfreien globalen Scan zu erlangen empfehle ich daher, jede einzelne Registrierung eines lokalen Scans zu bewerten und notfalls den lokalen Scan im globalen nicht weiterzuverwenden. Bei genügend überlappenden Punkten sollte dies zu keinen Schwierigkeiten führen.

Dies führt zur Beschreibung eines weiteren Problems, der richtigen Wahl der Position für einen aufzunehmenden Scan. Für die korrekte Funktion des ICP-Algorithmusses müssen genügend sich überlappende Punkte aus beiden Scans vorhanden sein. Der Abstand der Aufnahmepositionen der Scans darf also einen Maximum nicht überschreiten. In großen Räumen ist die Beachtung dieses Maximums auch kein Problem, schwierig wird es an Engstellen, z.B. einer Tür. Schießt man einen Scan wenige Meter vor der Tür, fährt dann durch diese hindurch und schießt den nächsten, so wird man keine Überlappungen der Scans feststellen. Eine auf 2D-Scans, die während der Fahrt aufgenommen werden, basie-

rende Abschätzung für geeignete 3D-Scanpositionen wäre also wünschenswert. Auf dieser Basis könnte man die Wahl der Scanposition auch automatisieren.

Hat man diese Automatisierung erreicht, könnte man beginnen, den aufgenommenen 3D-Scans eine Semantik zu geben. Im Moment werden diese Scans zwar erzeugt und zusammengefügt, die autonome Navigation des Roboters erfolgt aber nicht auf ihrer Grundlage. Allerdings wären für das Erkennen flacher oder von der Decke hinabragender Hindernisse gerade diese 3D-Scans geeignet. Um ihnen also eine Bedeutung zu geben, sprich, es werden für die Fahrt des Roboters freie Wege ermittelt, wäre eine Flächen- und Kantendetektion nötig. Waagrechte Flächen vor dem Roboter könnte man dann z.B. als befahrbar klassifizieren, soweit bis auf ihnen eine Kante erscheint.

Bei dieser Flächendetektion wird man zweifelsohne auf einige Probleme im globalen Scan im Bereich sich überlappender lokaler Scans stossen. Auch bei sehr guter Registrierung werden die dort vorhandenen Punkte beider Scans nie ganz aufeinander passen. Eine geeignete Filterung sollte hier also verwendet werden.

Auf dieses Problem stösst man unter anderem auch, will man für eine besser Visualisierung der Scans eine möglichst automatische Triangulierung durchführen. Für diese Triangulierung sollten zum Einen die im globalen Scan vorhandenen Punkte reduziert werden, zum Anderen nur geeignete Punkte gewählt werden. Schwierig wird dann das Verhalten an Stellen des Scans, an denen keine Messwerte vorhanden sind, Gründe hierfür könnten Schatten oder Objekte außerhalb der Reichweite des Scanners sein.

Hat man diese Triangulierung abgeschlossen und belegt die dadurch entstehenden Flächen mit Texturen, die von Kamerabildern stammen, die gleichzeitig mit der Aufnahme der 3D-Scans geschossen wurden, so kann eine äußerst realistische dreidimensionale Abbildung der Umgebung entstehen, in der man sich virtuell frei bewegen könnte. Für ein solches Zusammenfügen von Kamerabildern und 3D-Scans wäre es noch nötig, deren Koordinatensysteme ineinander zu transformieren und dann die Kamerabilder, basierend auf den Abstandsdaten, die der 3D-Scan liefert, entsprechend zu zerteilen. Eine solche Abbildung der Umgebung wäre warscheinlich sehr viel realistischer als künstlich erzeugte 3D-Modelle und möglicherweise auch sehr viel schneller und mit wesentlich geringerem Arbeitsaufwand zu generieren.

Anhang A

l3d-Spezifikation

A.1 Allgemeines

Das l3d-Format ist ein Dateiformat für 3D-Scans eines rotierbaren 2D-Laserscanners. Der Laserscanner wird während der Rotation nicht weiter bewegt und die Rotationsachse läuft durch seinen Mittelpunkt. Auf Grund dieser Gegebenheiten ist der gescannte Bereich eine Kugel und jeder Entfernungswert, den der Scanner liefert, kann Sphärenkoordinaten (θ , ϕ) zugeordnet werden. Um die Blickrichtung des Scans flexibel zu halten, wird zusätzlich die Drehrichtung des Laserscanners abgespeichert. Dies ermöglicht die korrekte Berechnung der sphärischen Koordinaten bei unterschiedlicher Drehrichtung.

A.2 Erweiterung

Um das gleiche Dateiformat verwenden zu können, ist das l3d-Format um ein karthesisches Koordinatensystem erweiter worden. Hierbei werden zum einen die Anzahl der Punkte sowie in drei float-Arrays die jeweiligen X-, Y- und Z-Anteile der Punkte gespeichert.

A.3 Sphärenkoordinaten

Ein Punkt $\mathsf{P} = (\boldsymbol{r}, \theta, \phi)$ wird in Kugelkoordinaten wie folgt beschrieben:

- $\boldsymbol{r}$ (Radius): Abstand vom Ursprung zum Punkt P. Da der Ursprung hier im Zentrum des Scanners steht, ist $\boldsymbol{r}$ gerade der Entfernungswert, den der Scanner zurückliefert.
- θ (Polarwinkel): Winkel zwischen positiver Z-Achse und P. Der Definitionsbereich von θ liegt normalerweise zwischen 0 und π, wir lassen hier aber einen Definitionsbereich von $-\infty$ bis $+\infty$ zu, um größere Flexibilität zu erhalten. Dadurch werden die Kugelkoordinaten allerdings nicht mehr eindeutig.
- ϕ (Azimutwinkel): Winkel zwischen positiver X-Achse und P_{xy} (P auf die XY-Ebene projeziert), gegen den Uhrzeigersinn gezählt. ϕ kann Werte zwischen $-\infty$ und $+\infty$ annehmen (analog zu θ)
- Drehrichtung: Achse um die sich der 2d-Scanner dreht (Achse des Azimutwinkels). Hierdurch wird nachher eine korrekte Umrechnung in ein karthesisches Koordinatensystem gewährleistet. Bisher implementierte Drehrichtungen:

  0 = Azimutachse nach oben (in Richtung Z-Achse des Roboters)

  1 = Azimutachse nach vorne (in Richtung X-Achse des Roboters)

A.4 Aufbau einer l3d-Datei

θ wird vom 3D-Scanner in N Schritten abgetastet, beginnend mit θ_0, ϕ wird in M Schritten abgetastet, beginnend mit ϕ_0. Der Scanbereich für θ heißt TR und der Scanbereich für ϕ heißt PR. Jedem Paar (θ_n, ϕ_m) wird mir r(θ_n, ϕ_m) ein Entfernungswert zugeordnet.

$$t_n = t_0 + n \cdot (TR/N),\ n \in \langle 0, \ldots, N \rangle \tag{A.1}$$

$$p_m = p_0 + m \cdot (PR/M),\ m \in \langle 0, \ldots, M \rangle \tag{A.2}$$

$\frac{TR}{N}$ entspricht der Auflösung des Scanners im Polarwinkel, $\frac{PR}{M}$ entspricht der Auflösung des Scanners im Azimutwinkel. TR und PR können negativ sein, um eine negative Rotation auszudrücken, z.B. wenn θ_0 45°entspricht und θ_1 40°(siehe Beispiel unten).

Eine Datei im l3d-Format ist dann wie folgt aufgebaut:

1. Format (4 Bits) :=
   - 0000 Standardformat (veraltet, Beschreibung der Drehachse des Scanners fehlt) ∨
   - 0001 Standartformat mit Blickrichtung des Scanners ∨
   - 0011 karthesisches Format der Punkte (hierbei werden die Werte zur Beschreibung des polaren und azimutale Scanbereiches auf 0 gesetzt) ∨
   - <andere> (nicht definiert)

WENN (Format == 0001) DANN

2. Beschreibung des polaren Scanbereichs (12 Byte) :=
   - θ_0 in Radiant (4 Byte float)
   - TR in Radiant (4 Byte float)
   - N (4 Byte integer)
3. Beschreibung des azimutalen Scanbereichs (12 Byte) :=
   - ϕ_0 in Radiant (4 Byte float)
   - PR in Radiant (4 Byte float)
   - M (4 Byte integer)
4. Beschreibung der Drehachse des Scanners (4 Byte) := Achsen-Richtung ((4 Byte integer)
5. Scandaten (N · M · 4 Byte (integer)) := r(θ_0, ϕ_0), r(θ_1, ϕ_0), r(θ_2, ϕ_0), . . ., r(θ_N, ϕ_0), r(θ_0, ϕ_1), r(θ_1, ϕ_1), r(θ_2, ϕ_1), . . ., r(θ_N, ϕ_1), . . ., r(θ_0, ϕ_M), r(θ_1, ϕ_M), r(θ_2, ϕ_M), . . ., r(θ_N, ϕ_M)

SONST WENN (Format == 0011) DANN

2. Punktanzahl (4 Byte) := Anzahl Punkte um Scan (4 Byte integer)

3. Scandaten:

   - Scandaten-X-Koordinaten (Punktanzahl · 4 Byte (float)) := x(0), x(1), ... , x(Punktanzahl - 1)
   - Scandaten-Y-Koordinaten (Punktanzahl · 4 Byte (float)) := y(0), y(1), ... , y(Punktanzahl - 1)
   - Scandaten-Z-Koordinaten (Punktanzahl · 4 Byte (float)) := z(0), z(1), ... , z(Punktanzahl - 1)

A.5 Beispiel

Der 2D-Scanner hat einen Scanbereich von 240° in 682 Schritten. Er ist nach oben ausgerichtet und dreht sich in 180 Schritten 180° um die Achse, die der Laserstrahl beim 341. Schritt bildet.
$t_0 = 120° = \frac{120}{180 \cdot \pi}$
$TR = -240° = \frac{-240}{180 \cdot \pi}$
$N = 682$
$p_0 = 270° = -90° = \frac{-\pi}{2}$
$PR = 180° = \pi$
$M = 180$

Anhang B

lsc-Spezifikation

B.1 Allgemeines

Zum Speichern einer 3D-Scanserie fungiert diese Datei als eine Art Playliste. Hierbei wird der jeweils eintreffende lokale Scan anhand seines Namens gespeichert. Ein eindeutiger Name wird ihm während des laufenden Betriebes anhand des Timestamp gegeben. Zusätzlich werden die zu dem lokalen Scan gehörenden Odometriedaten gespeichert um eine korrekte Registrierung mit Hilfe des ICP-Algorithmusses zu gewährleisten.

B.2 Aufbau einer lsc-Datei

Ein Scan wird samt seiner jeweiligen Daten in eine eigene Zeile geschrieben. Der Aufbau einer solchen Zeile ist wie folgt:

1. Laufende Nummer des jeweiligen Scans, beginnend bei 0 (integer)
2. .l3d-Dateiname des jeweiligen Scans (string)
3. Odometriedaten (6 * float) des jeweiligen Scans in der Reihenfolge x, y, z, Gierwinkel, Nickwinkel, Rollwinkel (Angabe der Winkel in Radiant)

B.3 Beispiel

Ausschnitt aus einer lsc-Datei:

```
0 30404.l3d 0 0 0 -0.001534 0 0
1 68015.l3d 0 0 0 -1.35452 0 0
2 137127.l3d 0 0 0 -2.40071 0 0
3 215853.l3d 2295.07 -736.661 0 -0.311402 0 0
4 378190.l3d 3190.9 69.6438 0 1.25167 0 0
```

Anhang C

CD-Inhalt

- LaTeXQuellcode
- Programmquellcode
  - Robbie-Programm (30_prog)
  - Puma-Quellcode (puma) (Spiegel der benötigten Pumainstallation)
- erzeugte Scandaten

Zum Kompilieren des Programms werden alle für das Robbie-Programm nötige Bibliotheken benötigt.

Will man nur das ICP-Programm verwenden genügt es, dass Robbie-Programm mit

```
./Robbie -c Config/ICP.conf
```

zu starten. Will man allerdings neue Scans aufnehmen so muss Robbie mit allen dafür nötigen Modulen gestartet werden.

Literaturverzeichnis

[AHB87] K. S. Arun, Thomas S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 9(5):698–700, 1987.

[Del07] Christian Delis. Entwicklung einer rotationsplattform für den hokuyo urg-04lx laserscanner. Studienarbeit, Universität Koblenz-Landau, 2007.

[ELF97] D. Eggert, Adele Lorusso, and R. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Mach. Vis. Appl*, 9(5/6):272–290, 1997.

[Nüc06] Andreas Nüchter. *Semantische dreidimensionale Karten für autonome mobile Roboter*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2006.

[PDMP06] Johannes Pellenz, Christian Delis, Ioannis Mihailidis, and Dietrich Paulus. Low-cost 3d-laserscanner für mobile systeme im robocup rescue wettbewerb. *9. Anwendungsbezogener Workshop zur Erfassung, Modellierung, Verarbeitung und Auswertung von 3D-Daten*, 2006.

[pro01] *3rd International Conference on 3D Digital Imaging and Modeling (3DIM 2001)*, Quebec City, Canada, 5 2001. IEEE Computer Society.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3rd International Conference on 3D Digital Imaging and Modeling (3DIM)* [pro01], pages 145–152.

[Wik06] Wikipedia. Kugelkoordinaten, 2006. Online: Stand 5. November 2006.

[WW03] O. Wulf and B. Wagner. Fast 3d-scanning methods for laser measurement systems. *14th International Conference on Control Systems and Computer Science*, 2003.