UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGAS

Entwicklung eines 3D Kartierungsalgorithmus für RoboCup Rescue

Bachelorarbeit
zur Erlangung des Grades
Bachelor of Science
im Studiengang Computervisualistik

vorgelegt von

Matthias von Steimker

Betreuer: Dipl.-Math. (FH) Dagmar Lang, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Dipl.-Math. (FH) Dagmar Lang, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im Mai 2011

Kurzfassung

Die RoboCup Rescue Liga wurde mit dem Ziel eines internationalen Austauschs zur Entwicklung von Rettungsrobotern gegründet. In Katastrophenregionen sollen diese Roboter verschüttete Opfer orten, deren gesundheitliche Verfassung erkennen und diese Informationen rechtzeitig an Rettungskräfte weitergeben.
An der Universität Koblenz wird seit mehreren Jahren der Rettungsroboter Robbie entwickelt. Über die gezielte Ansteuerung von Sensoren kann er Informationen über seine Umgebung sammeln und mit Hilfe dieser Informationen autonom in unbekannten Regionen agieren. Dafür erstellt Robbie eine 2D-Karte seiner Umgebung, um anhand dieser Karte navigieren und sich selbst lokalisieren zu können. Diese Karte stößt allerdings bei der Navigation über unebenes Gelände (wie z. B. Geröllhaufen) und spätestens in mehrschichtigen Katastrophengebieten auf ihre Grenzen, weswegen eine 3D-Kartierung benötigt wird.
Anhand des RoboCup Rescue Szenarios wird im Rahmen dieser Bachelorarbeit ein 3D-Kartierungsalgorithmus implementiert und hierfür die Probleme der 2D- und 3D-Kartierung ausführlich untersucht.

Abstract

The RoboCup Rescue League was founded with the intention to serve as an international communication platform for development of rescue robots. In regions hit by catastrophes, those robots are meant to find buried people, detect their physical condition and send the proper information to rescue teams.
At the university of Koblenz the rescue robot “Robbie“has been in development for years. Robbie accumulates information about his environment by targeted control of sensors and can act autonomous in unknown regions with help of the previous collected data. He creates an internal 2D map of his environment. This map provides enough information to navigate through space and to localize himself. Unfortunately, 2D maps have a huge drawback. When confronted with uneven terrain or even multilayered disaster areas, this technique will meet its limitations. Considered that most afflicted areas will probably have a bumpy ground, it is important to improve this technique. That is why 3D-mapping is being required. With the help of RobCup Rescue Scenario this Bachelor Thesis is going to implement a 3D-mapping algorithm and evaluate the flaws of 2D- and 3D mapping problems thoroughly.

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 Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten 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 23. Mai 2011

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Nicht erst seit dem Tōhoku-Erdbeben 2011 in Japan sind Rettungsroboter sehr gefragt um verschüttete Personen zu finden und deren Position und gesundheitliche Verfassung festzustellen, damit diesen schnellstmöglich geholfen werden kann.
So werden nun schon seit einigen Jahren Roboter für diese Ziele entwickelt und verbessert. Es ist abzusehen, dass die Bedeutung von Rettungsrobotern zunehmen wird, da man diese in Gebieten einsetzen kann, die für Menschen zu gefährlich sind und die Entwicklungen in der Robotik schnell voran gehen. So wurden gerade in den letzten Jahren bei der Entwicklung von 2D-Karten der Umgebung große Fortschritte erzielt, so dass nun die ersten Schritte in Richtung 3D-Karten folgen. Karten sind besonders wichtig, da potenzielle Opfer oder gefährliche Regionen in diesen markiert werden können und somit Rettungskräften ein genaueres Bild der Umgebung erlauben. Für autonome Roboter sind sie zusätzlich noch von Bedeutung, da sich der Roboter anhand einer Karte selbst in seiner Umgebung lokalisieren muss. Dies stellt bei zweidimensionalen Karten natürlich ein Problem dar, sobald das Einsatzgebiet über mehrere sich überschneidende Ebenen verteilt ist.
Aus diesem Grund ist der Schritt zur 3D-Karte sehr wichtig. Zusätzlich ermöglicht dieser natürlich ein noch genaueres Bild der Umgebung, da dreidimensionale Strukturen wie z. B. Menschen anhand ihrer Oberfläche einfacher erkannt werden können.

1.1 Aufbau der Arbeit

Die Arbeit ist folgendermaßen aufgebaut: In Kapitel 2 wird auf die Entwicklung und Zielsetzung der RoboCup Rescue Liga eingegangen und auf das grundlegende Problem der Kartierung. Darauf aufbauend stellt das folgende Kapitel 3 einige Verfahren zur Lösung dieses Problems vor. In Abschnitt 4 folgt eine tiefergehende Auseinandersetzung mit dem Problem der Kartierung, worauf die Umsetzung der

dreidimensionalen Kartierung, der Evaluation zweier Bibliotheken zur Nächsten Nachbar Suche, sowie zweier spezieller Registrierungsalgorithmen in Abschnitt 5 folgt. Schließlich wird im Kapitel 6 die Arbeit zusammengefasst und ein Ausblick auf offen gebliebene Problemfelder gegeben.

Kapitel 2

Grundlagen

2.1 RoboCup Rescue

Die RoboCup Rescue Liga entstand zusammen mit dem RoboCup 1997. Auslöser war das Erdbeben von Kōbe, welches 1995 etwa 6400 Todesopfer und 40.000 Verletzte forderte. Als Reaktion auf dieses Ereignis sollten Informationssysteme aufgebaut werden, die notwendige Informationen sammeln, verstärken, übertragen, zusammenfassen und verteilen können, umgehende Hilfe bei der Katastrophenhilfe leisten, sowie zuverlässig und robust während Routine- wie auch Notfallaufgaben arbeiten. [web11b]
Zur Bewältigung dieser Aufgaben sind heutige Roboter im Allgemeinen noch nicht ausgereift genug, so sagte Adam Jacoff vom US-amerikanischen National Institute of Standards and Technology (NIST), der die Liga maßgeblich mit aufbaute, in einem Interview am 1. April 2011: [web11a]

> „Bei realen Katastrophen haben wir dagegen noch keine Roboter im Einsatz gesehen und wir raten auch davon ab, denn die professionellen Retter sagen uns, dass sie eher stören als helfen."

Weiterhin sagt Jacoff, dass es ein

> „Ziel des Wettbewerbs [ist,] zunächst die besten Forschungsansätze zu identifizieren und in der Forschungsgemeinde bekannt zu machen, sodass das Feld insgesamt voran kommt"

sowie

> „Rettungskräfte mit Robotern vertraut zu machen und ihnen zu zeigen, was sie von Robotern erwarten können und was nicht."

Zum Erproben der Forschungsansätze entstand die RoboCup Rescue Arena. Diese simuliert eine Katastrophenumgebung mit verschiedenen Elementen wie beispielsweise unebenen Oberflächen, negativen Hindernissen (z. B. Klippen) oder den durch *stepfields* simulierten Schutthaufen (siehe Abb. 2.1).

Abbildung 2.1: Ein *stepfield*, welches einen Geröllhaufen simulieren soll. Die unterschiedlich hohen Blöcke erschweren es dem Roboter zu beurteilen, ob es sich hier um ein nicht befahrbares Hindernis handelt.

Die Arena ist dabei in durch Farben markierte Schwierigkeitsgrade unterteilt. Dabei symbolisiert die gelb den einfachen, orange den mittelschweren und rot den schweren Bereich. Der Schwierigkeitsgrad definiert die Art der Hindernisse und Barrieren die überwunden werden müssen. So gibt es im gelben Bereich nur leichte Schrägen und dieser Bereich ist recht gut zugänglich, während der Roboter im roten Bereich teilweise Treppen erklimmen muss.
Es gibt zwei große Aufgabenschwerpunkte während denen sich der Aufbau geringfügig ändert:

Bei der *rescue mission* müssen in einer begrenzten Zeit durch Puppen dargestellte, Opfer gefunden werden (siehe Abb. 2.2). Diese simulierten Opfer geben Wärme, CO_2 und akkustische Signale von sich. Für jedes gefundene Opfer werden Punkte vergeben.

Bei der *mapping mission* muss der Roboter in begrenzter Zeit so viel von der Arena erkunden wie möglich und gleichzeitig von der erkundeten Umgebung eine Karte erstellen. Auf dieser Karte müssen sogenannte *fiducials* (siehe Abb.2.2) möglichst gut erkennbar sein, da für jedes erkennbare *fiducial* Punkte vergeben werden.

Abbildung 2.2: Überblick über einen Teil der Arena auf der RoboCup German Open 2011. Im Hintergrund sind 15° Rampen zu erkennen. Im vorderen linken Abschnitt befindet sich ein *stepfield*. Neben der Treppe, die in der Mitte der Arena zu erkennen ist, befindet sich eine 30° Rampe mit einer anschließenden zweiten Ebene, auf der sich zwei durch Puppen simulierte Opfer befinden. Die blauen Tonnen (sogenannte *fiducials*), dienen der Qualitätsbewertung der erstellten Karte.

Zur Bewältigung der Aufgaben in der RoboCup Rescue Liga gibt es zwei verschiedene Herangehensweisen. Zum Einen mit teleoperiertem Roboter, welcher von einem menschlichen Operator gesteuert und kontrolliert wird und zum Anderen per autonomen Roboter. Bei dieser Methode muss der Roboter vollkommen autark die Aufgaben bewältigen. Es darf nur eingegriffen werden, wenn der Roboter sich selbst oder andere Roboter beschädigen bzw. Menschen verletzen könnte. Nach diesem Eingriff muss die Aufgabe jedoch von Neuem begonnen werden.
Im Rahmen dieser Bachelorarbeit wird ausschließlich der autonome Roboter Robbie (siehe Abb. 2.3) betrachtet. Im nächsten Teil wird näher auf diesen eingegangen.

2.2 Rettungsroboter Robbie

Die Entwicklung autonomer Roboter ist eine komplexe Aufgabe bei der es wichtig ist, dass Mechanik, Elektronik und Software zusammen gut funktionieren. Der Roboter muss dabei in der Lage sein, Daten von verschiedenen Sensoren zu empfangen, auszuwerten und mit der Umgebung entsprechend zu interagieren. [web11c]
An der Universität Koblenz arbeiten Computervisualistik- und Informatik-Studenten seit 2004 am Robbie-Projekt und entwickeln Hard- und Software in Projektpraktika konstant weiter.

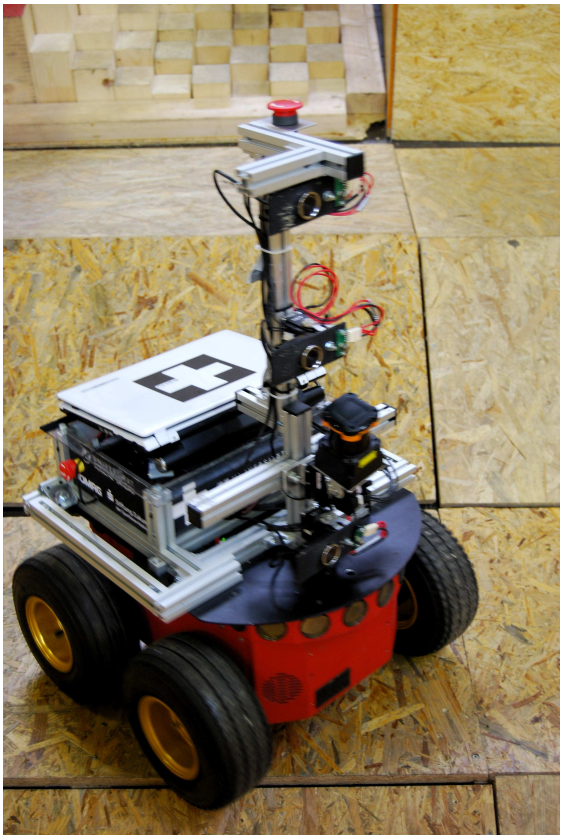


Abbildung 2.3: Robbie 16 während der RoboCup German Open 2011

Über verschiedene Sensoren kann Robbie seine Umgebung wahrnehmen. So besitzt die aktuelle Version 16 über Getriebe mit Odometrieerfassung, zwei nickbare 2D-Laserscanner, einen Lage- und Beschleunigungssensor (IMU, engl. *inertial measurement unit*), vordere und hintere Sonarsensoren, drei Webcams, sowie drei Wärmesensoren. Wie im vorherigen Abschnitt erwähnt, können während der Rettungsmission über die Wärmesensoren Opfer erkannt werden und sobald dies geschehen ist, werden die Webcams auf die Opferposition ausgerichtet. Für die Orientierung im Raum und zur Erstellung der Karten sind diese beiden Sensoren allerdings nicht von Bedeutung.

Der wichtigste Sensor hierfür ist der 2D-Laserscanner, mit dessen Hilfe Robbie die Abstände zu Objekten in seiner Umgebung innerhalb einer 2D-Ebene messen und sich damit überhaupt erst in seiner Umgebung orientieren kann. Der 2D-Laserscanner muss zur Abschätzung seiner Lage horizontal ausgerichtet werden, da es sonst zu Fehlmessungen kommen würde. Um dies zu erreichen ist auf dem Unterbau ein Lage- und Beschleunigungssensor montiert. Dieser misst die Pose,

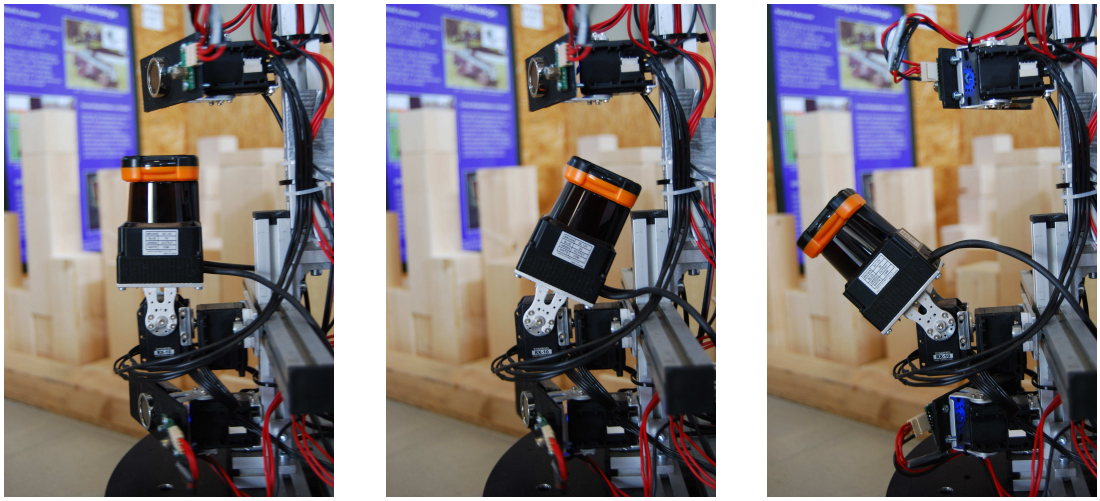
die Robbie im Raum einnimmt und mit dessen Hilfe der Laserscanner über einen Aktuator entsprechend ausgerichtet werden kann.
Weitere wichtige Informationen zur Bestimmung der Position im Raum bekommt Robbie über die Odometriemessung innerhalb des Getriebes. Die Odometrie Informationen können direkt über die API der Pioneer-Steuerungssoftware ARIA abgefragt werden. [Pel10] Mit Hilfe dieser kann Robbie abschätzen, um wie viel Grad er sich rotiert und welche Distanz er zurückgelegt hat.
Über die Erstellung der Karten anhand dieser Sensordaten, befasst sich der übernächste Abschnitt 2.4, vorher soll der 3D-Laserscanner, der zur Erstellung der 3D-Karten verwendet wird, beschrieben werden.

2.3 3D-Laserscanner

Zur Messung der Abstände zu Objekten in seiner näheren Umgebung (bis zu 5,60 Meter entfernt) ist an der Frontseite von Robbie der 2D-Laserscanner Hokuyo URG-30LX montiert.
Mit Hilfe dieses Sensors kann Robbie in einem Bereich von 240° vor sich, die Umgebungsoberflächen aktiv durch das Aussenden von Lichtstrahlen abtasten. Die Abtastrate beträgt dabei $0,36°$. Seit Robbie 16 ist es außerdem möglich einen zweiten 2D-Laserscanner einzusetzen, was einen vollen 360° Scan um Robbie herum ermöglichen würde.
Da es sich bei dem URG-30LX um einen 2D-Laserscanner handelt, ist dieser zur Aufnahme von 3D-Scans auf einem Dynamixel RX-10 Aktuator montiert.
Sobald das System einen 3D-Scan verlangt, wird der 2D-Laserscanner in die Startposition genickt und daraufhin in einer gleichmäßigen Bewegung nach vorne geneigt (siehe Abb. 2.4). Während dieser Bewegung werden kontinuierlich 2D-Scans aufgenommen und gespeichert. Sobald der Vorgang abgeschlossen ist, wird aus den aufgenommenen Daten eine 3D Punktwolke extrahiert. Die erreichbare vertikale Auflösung ist dabei abhängig von der Geschwindigkeit und Genauigkeit des Aktuators und der Frequenz des Laserscanners. [LHP+11]
Im folgenden Abschnitt wird näher auf den Aufbau des Scans eingegangen.

2.3.1 3D-Scan

Ein 3D-Scan besteht aus einer Menge von etwa 100.000 dreidimensionalen Punkten, die in einer Liste gespeichert werden. In Abb. 2.5 ist ein solcher 3D-Scan dargestellt. Der Vorgang des Scans dauert insgesamt 9 Sekunden. Während dieser Zeit muss der Roboter still stehen, gleichzeitig wird der 2D-Laserscanner dabei von oben nach unten genickt. Man kann sehr deutlich die durch die Nickbewegung entstehende, zeilenweise Aufnahme der Umgebung erkennen.



(a) Laserscanner in Ausgangsposition **(b)** Laserscanner in Startposition **(c)** Laserscanner in Endposition

Abbildung 2.4: Der Ablauf eines 3D-Scans. Im Ausgangszustand 2.4a ist der Laserscanner horizontal zum Erbboden ausgerichtet. Nach Anforderung eines 3D-Scans wird der Laserscanner nach hinten in die Startposition (2.4b) genickt. Danach wird der Laserscanner in gleichmäßiger Geschwindigkeit nach vorne geneigt, bis die Endposition für den 3D-Scan (2.4c) erreicht ist.

Für die Erstellung einer Karte ist es nun wichtig, verschiedene solcher 3D-Scans richtig zusammenzufügen. Dieses Problem wird im folgenden Abschnitt näher betrachtet.

2.4 Simultane Lokalisierung und Kartierung

Das wohl grundlegendste Problem der autonomen Robotik ist die simultane Lokalisierung und Kartierung des Roboters (SLAM, engl. *simultaneous localisation and mapping*).
Das Lokalisierungsproblem bezeichnet dabei das Problem der Bestimmung der Pose des Roboters im Raum. Dies geschieht anhand der erstellten Karte der Umgebung. Da aber die Umgebung für den Roboter unbekannt ist, muss diese Karte zuerst erstellt werden. Es muss somit eine Karte der Umgebung erstellt werden, während sich der Roboter in dieser gleichzeitig lokalisiert.
Die Lokalisierung geschieht über Sensordaten. Dabei kann die Position des Senders bekannt sein oder aufgrund der Sensordaten bestimmt werden. Ein beständiges Hinzufügen an Informationen über die Umgebung und updaten der Karte ist hierbei nötig (siehe Kapitel 3).

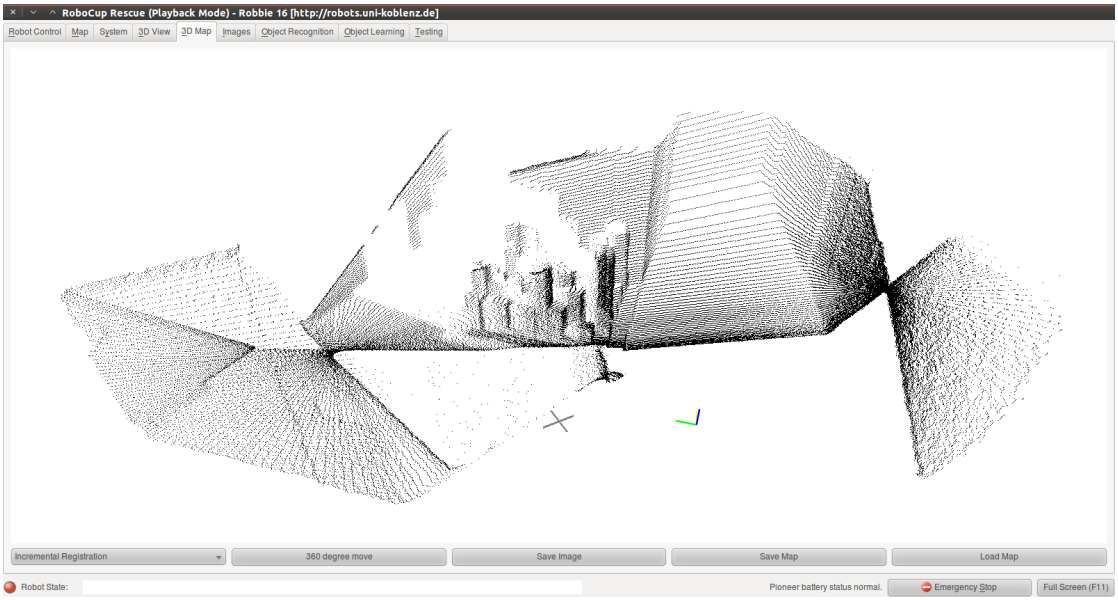


Abbildung 2.5: Die erzeugte Punktwolke eines einzelnen 3D-Scans, des auf einem Aktuator montierten Hokuyo URG-30LX Laserscanners im grafischen Benutzerinterface von Robbie 16.

Leider sind Sensordaten immer mit einem Fehler behaftet und unter bestimmten Umständen können sie ganz und gar falsche Informationen liefern. So erlauben es Odometriedaten abzuschätzen wie sich der Roboter gedreht und wohin er sich bewegt hat. Das Problem dabei ist allerdings, dass wenn der Roboter auf z. B. glatten Flächen rutschen sollte, die Odometriedaten nicht mehr mit der tatsächlichen Rotation und Translation des Roboters übereinstimmen. Damit würden auch zwei verschiedene Scans nicht in einer Karte zusammengefügt werden können.

Dies ist ein Grund warum ausgeklügelte Matchingverfahren entwickelt werden, um trotz falscher Odometrie eine möglichst genaue Karte der Umgebung erstellen zu können. Für diese Methoden gibt es einige verschiedene Ansätze, z. B. über Extended Kalman-Filter, über Partikelfilter, Expectation-Maximization-Filter oder Graph-basierte Techniken. Auf einige dieser Methoden wird im Kapitel 3 noch genauer eingegangen.

Ein Schwerpunkt dieser Arbeit befasst sich mit der Lösung des simultanen Lokalisierungs- und Kartierungsproblems im dreidimensionalen Raum, d. h. unter Berücksichtigung der Postion des Roboters im dreidimensionalen Raum sowie seiner Orientierung.

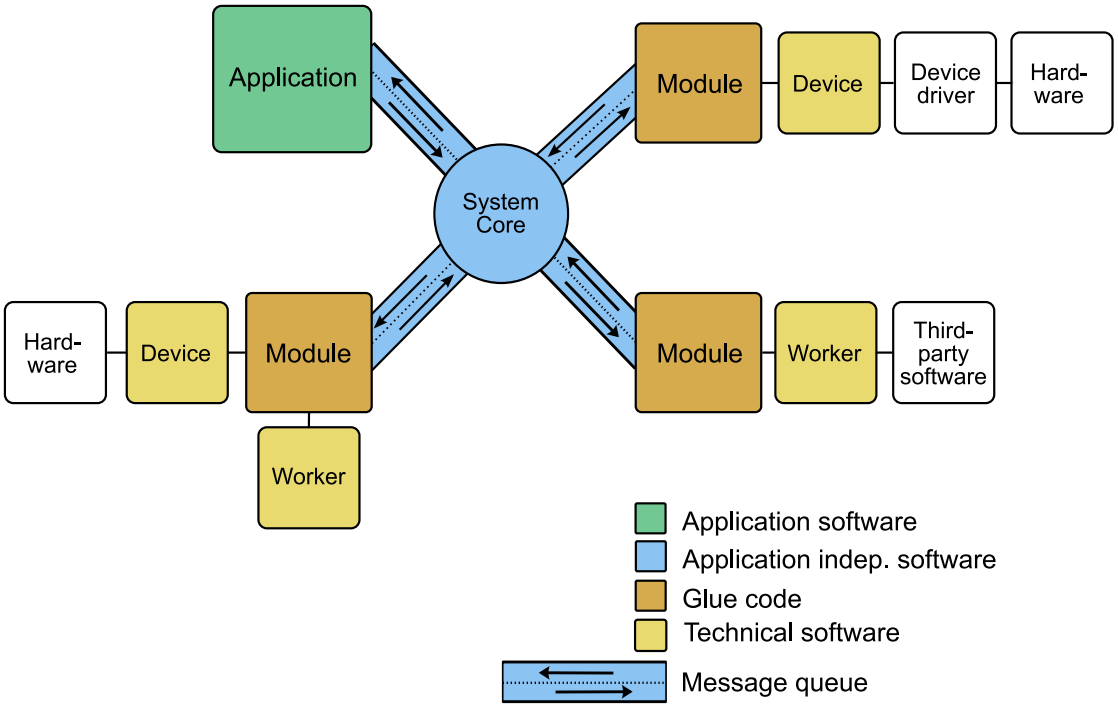


Abbildung 2.6: Die Robbie zugrundeliegende Softwarearchitektur. (Quelle: [TSPP11])

2.5 Softwarearchitektur von Robbie 16

Der Software von Robbie 16 liegt ein Nachrichtensystem zu Grunde, d. h. *Modules* kommunizieren mit Hilfe von *Messages* miteinander. Dabei stecken die eigentlichen Algorithmen in den *Workern*. Ein *Module*, wie beispielsweise das *Slam3DModule* abonniert *Messages* und erhält über diese *Messages* Daten, welche über inkludierte *Worker* verarbeitet oder ausgegeben werden. Hardwarekomponenten werden dabei über *Devices* angebunden. Nachfolgend wird die Robbie zugrundeliegende Softwarearchitektur nach [TSPP11] beschrieben und in 2.6 schematisch abgebildet.

Die *Messages* werden vom *Core* verwaltet. Dieser ist unabhängig von den Applikationen und der Hardware und dient als eine Art *Message Router* mit einer *In-* und *Outbox* in denen eingehende bzw. ausgehende Nachrichten landen. Die Module müssen nicht im Programmcode definiert werden, sondern werden in einer Konfigurationsdatei gelistet und nach dem Start automatisch mit dem *Core* registriert. Dieser leitet dann an alle registrierten *Modules* die jeweils abonnierten *Messages* weiter. Die Module übernehmen die Kommunikation der einzelnen Komponenten der Software und dienen dabei als eine Schicht zwischen den *Workern* und den *Devices*. Die Kommunikation erfolgt über das Lesen und Erstellen der *Messages*. Der Vorteil dabei ist, dass die *Devices* und *Worker* nicht mit systemspezifischen

Objekten umgehen müssen und dadurch unabhängig vom System bleiben. Dabei gibt es zwei Arten von *Modules*: Die passiven *Modules* werden erst durch Ankunft einer *Message* aktiviert, während die aktiven *Modules* zusätzlich in festgelegten Zeitintervallen aufgerufen werden.
Die *Devices* sind dafür verantwortlich die Hardware anzusteuern, das heißt Daten zu lesen und Funktionalität zu kontrollieren. Dabei können *Devices* auf zwei verschiedene Arten mit der Hardware verbunden werden: Zum Einen als eine extra Schicht für externe Hardwaretreiber und zum Anderen können sie die Hardware direkt anbinden.
Die *Worker* sind die Arbeitstiere der Software. Ein *Worker* wird für eine bestimmte Aufgabe erstellt und in ihm laufen die eigentlichen Algorithmen zur Verarbeitung der Daten ab. Wie auch Devices, können *Worker* an externe Bibliotheken angebunden werden und benötigen kein spezifisches Wissen über die *Modules* oder *Messages*.
Zwei Instanzen des *Frameworks* können auch über eine Client-Server-Verbindung miteinander verbunden werden. Dabei läuft der Server auf dem Roboter. Der Vorteil besteht darin, dass aufwendige Berechnungen, deren Ergebnisse nicht sofort vom Roboter benötigt werden, auf dem Client ausgeführt werden können.

2.5.1 Szenengraph

Der Szenengraph ist eine Datenstruktur zur computergrafischen Darstellung der dreidimensionalen Szene. Diese Datenstruktur ist als eine Baumstruktur aufgebaut, bei der die Wurzel als das Weltkoordinatensystem und jeder weitere Knoten als eigenes lokales Koordinatensystem verstanden werden kann. [Mül11]
In den Knoten befinden sich dabei die einzelnen Objekte der Umgebung. Beispielsweise wurde Robbie für den Szenengraphen modelliert und befindet sich im Roboterkoordinatensystem. Zusätzlich wurden die Sensoren, die Robbie zur Verfügung stehen, ebenfalls als Unterbaum von Robbie modelliert und befinden sich in ihren jeweiligen Sensorkoordinatensystemen.
Dadurch, dass die Transformationen der jeweiligen Teilbäume ebenfalls gespeichert werden, ist durch das Traversieren des Baumes ein einfaches „positionieren“ der lokalen Koordinatensysteme in der Welt möglich. [Mül11]
Der Szenengraph hat für die 3D Kartierung insofern eine Relevanz, da aus diesem die Positionsänderungen von Robbie ausgelesen werden können. In Kapitel 5 wird auf den Szenengraphen noch einmal Bezug genommen.

Kapitel 3

Stand der Technik

1 In diesem Teil der Arbeit sollen vor allem verschiedene Ansätze zur Lösung des SLAM-Problems vorgestellt werden.
Holz und Behnke zeigten in [HB10] verschiedene Algorithmen zur Reduktion der Punkte eines Scans oder der Qualitätssteigerung der berechneten Karte auf. Diese Algorithmen sollen hier ebenfalls vorgestellt werden, da die Punktreduktion für den Metascan, auf den im Abschnitt 4.3.2 näher eingegangen wird, von Bedeutung ist. Im letzten Abschnitt des Teils dieser Arbeit wird untersucht, wie andere RoboCup Rescue Teams 3D-Karten erstellen.

3.1 Lösungsansätze für das SLAM Problem

Das SLAM Problem gilt als eines der grundlegendsten der Robotik. Aufgrund dessen gibt es einige verschiedene Ansätze. Grob lassen sich diese in die zwei Gruppen probabilistisch und Graph-basiert unterteilen. Die Einteilung der Algorithmen, sowie die Vorstellung der Verfahren im folgenden Abschnitt ist aus Pellenz [Pel10] entnommen.

3.1.1 Probabilistische SLAM Algorithmen

Probabilistische SLAM Algorithmen behandeln explizite Unsicherheiten über das „geführte Abschätzen“ und die verarbeiteten sensorischen Informationen. Dabei wird die Wahrscheinlichkeitsverteilung über alle möglichen Lösungen abgeschätzt. Der Vorteil dieser Verfahren ist die hohe Robustheit und Genauigkeit der Daten. Allerdings ist der Berechnungsaufwand probabilistischer Ansätze zu hoch zum Erstellen dreidimensionaler Karten. Dennoch soll hier ein kurzer Überblick über gängige Verfahren gegeben werden.

Der Erweiterte Kalman-Filter (EKF) ist, anders als der Standard Kalmanfilter, auch für Systeme mit nicht linearem Verhalten ausgelegt. Die Karte ist bei diesem Verfahren merkmalsbasiert und durch punktförmige Landmarken (Säulen, Ecken, etc.) definiert. Dabei werden relativ wenige Landmarken verwendet ($<$1000) und es können auch nur positive Sichtungen von Landmarken bearbeitet werden. Der Roboter schätzt die Pose des Roboters und zusätzlich die Positionen aller Landmarken, die der Roboter zu sehen bekommen hat. Die Zahl der Landmarken nimmt dabei natürlich immer weiter zu.

Partikelfilter stellen ebenfalls eine gängige Lösung dar. Bei dieser wird der geschätzte Zustand des Systems durch eine endliche Anzahl von Werten, die bestimmten Gebieten im Zustandsraum entsprechen, repräsentiert. Der Vorteil des Partikelfilters ist, dass dieser nicht auf bestimmte Funktionen (wie beispielsweise der Gaußfunktion) zur Repräsentation des aktuellen Zustands angewiesen ist. Robbie verwendet zur 2D-Kartierung Partikelfilter, da sich dieses Verfahren als sehr stabil erwiesen hat.

Der Expectation-Maximization-Algorithmus (EM-Algorithmus) ist ein weiteres vielseitig eingesetztes Verfahren und beschreibt die Schätzung von Parametern statistischer Modelle. Der Algorithmus ist in zwei Schritte unterteilt. Bei dem *Expectation*-Schritt wird angenommen, dass die Karte bekannt ist und der Pfad des Roboters anhand dieser Karte, der Messungen und der Aktionen geplant wird. Bei dem *Maximization* wird dagegen umgekehrt der Pfad als bekannt angenommen und die wahrscheinlichste Karte bestimmt. Da der Rechenaufwand dieser Lösung sehr hoch ist, kann diese lediglich als offline Lösung verwendet werden.

3.1.2 Graph-basierte SLAM Algorithmen

Graph-basierte Algorithmen zur Lösung des SLAM-Problems nutzen Baumstrukturen zur Speicherung der Laserscans und Robter-Posen.
Beispielsweise kann in den Knoten die Roboter-Pose oder ein Laserscan gespeichert werden, während in den Kanten zwischen zwei Knoten die Messwerte, die auf einer räumlichen Beschränkung basieren, abgelegt werden. Ein Optimierungsverfahren kann dabei die wahrscheinlichste Konfiguration der Knoten bestimmen, wenn der Graph aufgebaut ist. Danach kann die Karte generiert werden.
Es gibt viele verschiedene Varianten der Graph-basierten Lösung und da diese Lösung auch für die Lösung des dreidimensionalen SLAM Problems geeignet ist, nutzt z. B. Robbie für die Speicherung der 3D-Scans ein Verfahren, bei dem die 3D-Scans in den Knoten und die Pose-Änderungen in doppelten Kanten gespeichert werden.

3.2 Datenreduktion und Qualitätssteigerung der Karte

Holz und Behnke [HB10] beschreiben verschiedene Verfahren zur Punktreduktion eines 3D-Scans. Diese Verfahren haben vor allem das Ziel, die Qualität der Karte zu steigern indem zum Beispiel komplette Scans oder einzelne Punkte verworfen werden. Die Verfahren wurden von Holz und Behnke in Zusammenhang mit dem zweidimensionalen SLAM Problem vorgestellt.
Die Punktreduktion ist besonders für den Metascan, auf den in Abschnitt 4.3.2 noch näher eingegangen wird, von Bedeutung. Der Grund ist, dass diesem Scan alle Punkte eines neu generierten Scans hinzugefügt werden. Eine Qualitätssteigerung der Karte ist im Allgemeinen immer sinnvoll. Folgend, sollen vier von Holz und Behnke vorgestellte Verfahren aufgezeigt werden:

sparse point maps stellen eine erste hier vorgestellte Möglichkeit der Punktreduktion dar. Die Schlüsselidee dieses Verfahren liegt in der Vermeidung der Speicherung doppelter Punkte, die zu einem gleichen Punkt in der realen Umgebung korrespondieren. Dieses Verfahren wurde im Rahmen dieser Bachelorarbeit implementiert. Im Abschnitt 4.4 wird dieses Verfahren genauer erklärt während im Abschnitt 5.3.2 die Implementation und Ergebnisse evaluiert werden.

Verschiedene heuristische Ansätze bieten sich ebenfalls zur Punktreduktion an. Im Vergleich zu den sparse point maps können diese die Qualität der Karte steigern. Beispielsweise können die Odometrieänderungen überwacht werden und falls diese zu große invalide Sprünge aufweisen, kann der aktuelle Scan verworfen werden oder mit geringerer Gewichtung in den Metascan einfließen. Die Idee dahinter ist, dass die Odometrie gerade dann falsche Daten liefert, wenn der Roboter z. B. ins Rutschen gekommen ist und diese falschen Daten keine negativen Auswirkungen mehr auf den Metascan hätten. Größere Odometriefehler haben somit keine negativen Auswirkungen auf den Metascan.

Die Anzahl korrespondierender Punkte zwischen dem neuesten und dem Metascan kann ebenfalls untersucht werden. Sollte dabei das Verhältnis zur Gesamtpunktzahl des Scans unter einem bestimmten Schwellwert liegen (z. B. unter 30%), dann kann dieser Scan ebenfalls verworfen werden. Dieses Vorgehen macht insofern Sinn, als das es bei geeignet hoher Frequenz der Scans genügend Überlappungen geben muss.

Die Modifizierung des ICP-Algorithmus stellt eine weitere von Holz und Behnke beschriebene Möglichkeit der Punktreduktion dar. Während des-

sen Berechnung könnten einzelne korrespondierende Punkte dessen Punkt-zu-Punkt Distanz über einem festgelegten Schwellwert liegen, ebenfalls verworfen werden. Ein Wert diesem Schwellwert könnte dabei schlechte initiale Abschätzungen, aber dafür eher falsche Korrespondenzen, im Vergleich zu einem niedrigeren Schwellwert, erfassen. Den Schwellwert kann man dabei auch dynamisch wählen und exponentiell sinken lassen. Ergebnis sollte hierbei wiederum sein, dass größere Registrierungsfehler von falschen Odometrieabschätzungen oder Laserscans vermieden werden könnten.

3.3 Vorgehen anderer Rescue Teams bei der Erstellung von 3D-Karten

Auf Grund der großen unerschlossenen Potenziale einer 3D-Karte forschen auch andere RoboCup Rescue Teams an der Entwicklung eines 3D-Kartierungsalgorithmus. Das Vorgehen zwei dieser Teams soll hier beschrieben werden.

3.3.1 Team CASualty (Australien)

Das Team CASualty aus Australien[MMS+10] nutzt den FastSLAM Algorithmus [Mon03] zur Erstellung von Karten in Form von Belegtheitskarten (engl. *occupancy grid maps*) um über diese die Roboterposition zu korrigieren und die Karte zu aktualisieren.
Die occupancy grid maps sind in Zellen unterteilt und jede einzelne Zelle enthält eine Höheninformation. Eine dreidimensionale Karte der Umgebung soll dabei ausreichen um eine einzige Ebene zu repräsentieren. Die Karte unterstützt dabei die erwarteten Sensordaten welche vom FastSLAM Algorithmus benötigt werden, während notwendige Kartenaktualisierungen effizient ausgeführt werden. Wenn das dreidimensionale Positions-Tracking verfügbar ist, wird die Karte mit zusätzlichen Höheninformationen erweitert um eine echte 3D Karte mit unterschiedlichen Ebenen zu erhalten.
Durch das Separieren des Positions-Tracking Algorithmus vom Rest des FastSLAM Algorithmus wird eine verteilte Implementierung möglich. Weil die SLAM Lösung einen signifikanten Anteil der Verarbeitung benötigt, wird der Algorithmus auf einem *remote* Computer berechnet, während das Positionstracking vom Computer des Roboters selbst erledigt wird. Somit kann der Roboter für kurze Zeit seine eigene Position bestimmen, während er periodische Korrekturen von der Basisstation erhält. Dies ermöglicht dem Roboter temporär unabhängig von der Basisstation zu agieren, falls die Verbindung zu dieser nicht verfügbar ist. Der FastSLAM Algorithmus generiert für alle vorgeschlagenen Pfade eine *occupancy grid heightmap*. Nach dem Lauf wird der wahrscheinlichste Pfad ausgewählt und eine Höhe sowie

ein Schwellwert für die Wahrscheinlichkeit hinzugefügt um ein zweidimensionales *occpupany grid* zu erstellen.

3.3.2 Team Pelican United (Japan)

Das Team Pelican United aus Japan [OY09] verwendet zur Erstellung von 3D-Karten zwei 2D-Laserscanner. Einer ist dabei horizontal und der andere vertikal ausgerichtet. Wie bei Robbie wird über den eingebauten Lage- und Beschleunigungssensor die Orientierung im Raum und über Odometriemessungen die Translation geschätzt. Das Matching erfolgt mit Hilfe eines 3D-SLAM Algorithmus über die Informationen des horizontal ausgerichteten 2D-Laserscanners, bei gleichzeitiger Nachkorrektur vom Operator.

Kapitel 4

Mapping

In Kapitel 2 sowie 3 wurde schon auf das SLAM-Problem und verschiedener Lösungsansätze sowie Ansätzen zur Punktreduktion und Qualitätssteigerung der Karten eingegangen.
In diesem Kapitel soll nun der gesamte Vorgang der Kartierung (eng. *mapping*) erläutert werden. Als Kartierung wird in diesem Sinne das Erstellen einer Karte der Umgebung bezeichnet. Ein besonderes Augenmerk liegt in dieser Arbeit auf der Erstellung von 3D-Karten. Dafür wird die 3D-Pose des Roboters in der Umgebung benötigt. Die Pose wird durch eine dreidimensionale kartesische Koordinate, sowie drei Euler-Winkel beschrieben, was die 6 Freiheitsgrade $(x, y, z, \theta_x, \theta_y, \theta_z)$ ergibt, weswegen die Lösung des SLAM-Problems im dreidimensionalen Raum auch als 6D-SLAM bezeichnet wird (D steht hier für Freiheitsgrad (engl. *degree of freedom*)).
Um Karten zu erstellen, müssen verschiedene Momentaufnahmen der Umgebung zu einem Gesamtbild zusammengesetzt werden.
Diese Momentaufnahmen sind in unserem Fall die 3D-Scans der Umgebung. Die verschiedenen 3D-Scans müssen miteinander registriert, d. h. in ein gleiches Koordinatensystem gebracht werden, damit eine einheitliche Karte entstehen kann. Wie schon in Abschnitt 2.4 erklärt, sind Sensordaten fehlerbehaftet. Aus diesem Grund reicht es nicht aus, einen Scan lediglich in das Koordinatensystem eines anderen Scans zu führen. Es muss ein entsprechendes Scanmatching Verfahren angewendet werden. Die Scanmatching Verfahren nutzen im Detail zur Registrierung neuer Scans den iterativen Algorithmus der nächsten Punkte (ICP-Algorithmus, engl. *iterative closest point algorithm*) um einen neuen Scan richtig zu registrieren. Der ICP-Algorithmus sucht dabei korrespondierende Punkte zwischen zwei Scans. Dafür ist eine schnelle nächste Nachbar Suche unabdingbar.

In diesem Kapitel der Bachelorarbeit wird zunächst auf die nächste Nachbar Suche eingegangen und die zwei Bibliotheken ANN und FLANN, die dieses Problem schnell lösen können, vorgestellt. Danach wird der ICP-Algorithmus im Detail

erklärt, woraufhin die zwei Scanmatching Methoden der paarweisen Registrierung und das Metascanmatching vorgestellt werden. Als letzter Punkt dieses Kapitels folgt eine Beschreibung des implementierten Datenreduktionsverfahren der *sparse point maps*.

4.1 Nächste Nachbar Suche

Die nächste Nachbar Suche ist ein häufiges Problem in der Informatik. Man stelle sich beispielsweise eine semantische Karte eines Stadtteils vor und möchte untersuchen zu welchem Gebäude welcher Hydrant am nächsten liegt. Dieses Problem lässt sich recht einfach lösen, indem für jedes Haus in der Karte die Distanzen zu allen Hydranten auf der Karte untersucht werden und jeweils die kürzeste Distanz gespeichert wird. Im Endeffekt würde dies ein quadratisches Problem bedeuten.

Im Fall des 6D-SLAM reicht diese Lösung allerdings nicht aus, da hier für einige tausend Punkte ein nächster Nachbar gefunden werden muss und das bevor der nächste 3D-Scan erzeugt wurde.

Aus diesem Grund wird eine schnellere Methode benötigt. Baumstrukturen wie der k-d-Baum haben sich dabei für die nächste Nachbar Suche als besonders geeignet erwiesen.

k-d-Bäume basieren auf einer rekursiven Unterteilung des Raums in disjunkte Hyperrechtecke, den sogenannten Zellen. Diese Hyperrechtecke entsprechen im zweidimensionalen Fall einem Rechteck und im dreidimensionalen Fall einem Quader. Analog verhält es sich im mehrdimensionalen Raum. Jeder Knoten des Baumes ist mit solch einem Hyperrechteck assoziiert und damit auch mit den Punkten die in diesem liegen.

Der Wurzelknoten umfasst den gesamten Raum und damit alle Punkte. Bei der Aufstellung des Baumes, werden die entstehenden Hyperrechtecke solange durch eine Splitebene getrennt, bis die Anzahl an Punkten im jeweiligen Hyperrechteck unter einer Maximalanzahl (der sogenannten *bucket size*) liegt. Hierbei werden die Punkte, je nachdem auf welcher Seite der Splitebene sie liegen, in das entsprechende Hyperrechteck eingeordnet.

Mit den beiden entstandenen Hyperrechtecken werden wiederum neue Knoten assoziiert und diese an den Knoten des ursprünglichen Hyperrechtecks angehangen. Wenn die Anzahl der Punkte in dem Hyperrechteck unter die *bucket size* fällt, wird der entsprechende Knoten als Blattknoten markiert und in diesem entsprechend die Punkte gespeichert.

4.1.1 Approximate Nearest Neighbor

ANN (engl. *Approximate Nearest Neighbor*) ist eine Bibliothek zum Suchen des nächsten Nachbarn in verschiedenen Dimensionen [Mou10]. Dabei wird die exakte, sowie die approximierte Suche unterstützt. Entwickelt wurde diese Bibliothek von David M. Mount von der University of Maryland und Sunil Arya von der Hong Kong University of Science and Technology. ANN unterstützt zwei Datenstrukturen: *k-dimensional trees* (k-d-Bäume) und *box-decomposition trees* (b-d-Bäume). [Mou10]
Der b-d-Baum ist von Vorteil wenn Punkte sehr gehäuft auftreten, da diese ein Problem bei der Partitionierung darstellen können. So könnten beim k-d-Baum sehr langgezogene Hyperrechtecke entstehen, in denen die Suche ineffizient ist. Die b-d-Bäume bieten hierbei eine größere Robustheit. Der Hauptunterschied zum k-d-Baum liegt dabei in der zusätzlichen Zerlegungsoperation, dem sogenannten Zusammenziehen (v. engl. *shrinking*).

4.1.2 Fast Library for Approximate Nearest Neighbors

Die von Marius Muja und David G. Lowe von der University of British Columbia entwickelte Bibliothek FLANN (engl. *Fast Library for Approximate Nearest Neighbors*) ist ebenso wie ANN für die Suche nach nächsten Nachbarn in höherdimensionalen Räumen optimiert worden. Aufgrund dieser Optimierung nutzt FLANN keinen klassischen k-d-Baum, sondern mehrere zufällige k-d-Bäume. Bei diesen zufälligen k-d-Bäumen wird die Aufteilungsdimension zufällig durch Auswahl der ersten d Dimensionen, bei denen die Daten die größte Varianz haben, gewählt. d ist dabei fest auf 5 eingestellt. Andere Parameter können von FLANN ebenfalls automatisch anhand der gegebenen Daten konfiguriert werden, um eine möglichst hohe Performanz zu erzielen.
Der Grad der Approximation ist festgelegt durch Begutachten einer festen Anzahl von Blattknoten des k-d-Baums, wo die Suche beendet wird und die besten Kandidaten zurückgegeben werden. [Muj09]

4.2 Iterative Closest Point Algorithmus

Der *Iterative Closest Point* Algorithmus wurde 1991 zur Registrierung zweier Punktmengen M (Modellmenge) und D (Datenmenge) beschrieben und entwickelt [BM92]. Dabei werden iterativ korrespondierende Punkte zwischen den beiden Punktmengen gesucht und durch Anpassung der Transformationsparameter immer weiter verfeinert bis die Kostenfunktion $E(\boldsymbol{R}, \boldsymbol{t})$ minimal ist, oder ein anderes Terminierungskriterium auftritt. ($\boldsymbol{R}$,$\boldsymbol{t}$) ist dabei die Transformation zwischen den beiden Scans, mit der Rotationmatrix $\boldsymbol{R}$ und dem Translationsvektor $\boldsymbol{t}$.

Die Kostenfunktion $E(\boldsymbol{R}, \boldsymbol{t})$ ist nach [Nüc06] wie folgt definiert:

$$E(\boldsymbol{R}, \boldsymbol{t}) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} \|\boldsymbol{m_i} - (\boldsymbol{R}\boldsymbol{d_j} + \boldsymbol{t})\|^2 \tag{4.1}$$

Hierbei sind $\boldsymbol{m}_i$ und $\boldsymbol{d}_j$ mit $i \in \mathbb{N}, 1 \leq i \leq N_m$ und $j \in \mathbb{N}, 1 \leq j \leq N_d$ die Punkte der Modellmenge M bzw. der Datenmenge D. N_m und N_d ist die Anzahl der Punkte in M bzw. D. $w_{i,j}$ ist ein Gewicht, welchem der Wert 1 zugewiesen wird, wenn $\boldsymbol{m}_i$ und $\boldsymbol{d}_j$ zum selben physischen Punkt in der Umgebung korrespondiert. Ansonsten ist $w_{i,j}$ 0.
Zur Berechnung der einzelnen Punktkorrespondenzen wird die Summe der Quadrate des Abstandes zwischen $\boldsymbol{m}_i$ und dem transformierten $\boldsymbol{d}_j$ berechnet und dies so lange iterativ wiederholt, bis die Differenz des durchschnittlichen quadratischen Fehlers unter einem vorher definierten Schwellwert ϵ liegt.
Für M und D betrachtet, geht der ICP-Algorithmus nach [Nüc06] folgendermaßen vor:

1. Suche für jeden Punkt $\boldsymbol{d}_i \in D$ den nächsten gelegenen Punkt in M.
2. Bestimme aus den ermittelten korrespondierenden Punkten die Transformation $(\boldsymbol{R}, \boldsymbol{t})$, die die Kostenfunktion $E(\boldsymbol{R}, \boldsymbol{t})$ minimiert.
3. Wende die gefundene Transformation auf die Punktmenge D an.
4. Minimiere die Kostenfunktion $\boldsymbol{E}$ oder halte an, wenn ein Terminierungskriterium zutrifft

Zur Minimierung der Kostenfunktion existieren die zwei Prinzipien der direkten und indirekten Verfahren.

Die direkten Verfahren haben gemeinsam, dass sie Rotation und Translation getrennt voneinander betrachten. Dabei sind derzeit vier Algorithmen die die Kostenfunktion des ICP-Algorithmus in geschlossener Form minimieren, bekannt:

- Transformationsschätzung mittels der Singulärwertzerlegung einer Matrix [AHB87]
- Transformationsschätzung mit Hilfe von Orthonormal-Matrizen [HHN88]
- Transformationsschätzung unter Verwendung des Einheitsquaternion [Hor87]
- Transformationsschätzung mit Dualquaternion [WSV91]

Im Rahmen dieser Bachelorarbeit wurde auf die Transformationsschätzung mit Hilfe von Dualquaternionen zurückgegriffen.

Den indirekten Verfahren werden diejenigen hinzugezählt, die beispielsweise ein physikalisches Federsystem [EFF98, SH96] simulieren. Bei diesem werden die in der Kostenfunktion aufsummierten Abstände als Federn betrachtet, die die zu transformierende Punktmenge in Richtung der anderen bewegt. Es gibt noch andere indirekte Verfahren, allerdings haben diese alle den Nachteil gemeinsam, dass sie für die Berechnung der Transformationen an verschiedenen Stellen die Auswertung der Kostenfunktion benötigen, wodurch diese zwangsläufig rechenintensiver als die direkten Verfahren sind. [Nüc06]

Es wurden verschiedene Variationen und Erweiterungen des ICP-Algorithmus entwickelt. Diese sollen Effizienz und Robustheit verbessern und beziehen sich immer auf die Metrik der gemessenen Distanzen, der Minimierung der Kostenfunktion $E(\boldsymbol{R}, \boldsymbol{t})$, dem Gewichten der Korrespondenzen. [HB10]

4.3 Varianten der Registrierung

Um zwei Punktmengen M und D miteinander zu registrieren, wurden von Holz und Behnke [HB10] zwei verschiedene Varianten der Registrierung der neuesten Punktmenge D zu M aufgezeigt. Die beiden Verfahren unterscheiden sich dabei in der Auswahl von M.
Im Kapitel 5 werden diese beiden Verfahren untersucht und die Ergebnisse visuell evaluiert.

4.3.1 Paarweise Registrierung

Die paarweise Registrierung des neuesten Scans D gegen den vorherigen M ist die einfachste Form der Registrierung eines neuen Scans. Bei diesem Verfahren werden die Änderungen der Orientierung und der Pose zwischen den beiden Scans ermittelt (siehe Abb. 4.1). Da dabei nur zwei Scans betrachtet werden, ist dieses Verfahren sehr effizient, jedoch akkumulieren sich Registrierungsfehler dabei sehr stark, was letztendlich zu Inkonsistenzen in der Karte führt.

Abbildung 4.1: Paarweise Registrierung

Die Paarweise Registrierung ist dabei insofern erweiterbar, dass iterativ solange der jeweilige Scan mit dem vorherigen registriert wird, bis sich die berechnete Pose nicht mehr verändert (siehe Abb. 4.2). Dadurch wird die Komplexität der Berechnung stark erhöht, allerdings können aufgrund der hinzugewonnenen Informationen Inkonsistenzen ausgeglichen werden.

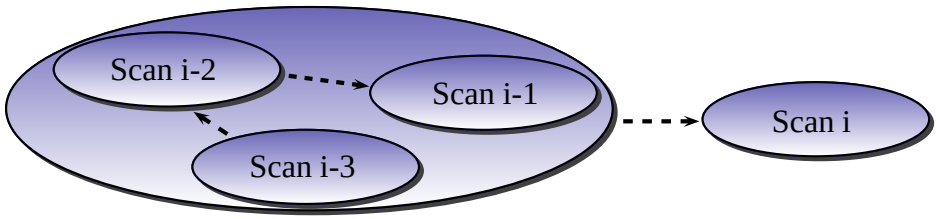


Abbildung 4.2: Paarweise Registrierung II

4.3.2 Metascanmatching

Eine weitere Möglichkeit ist das Matching mit einem Metascan. Der Metascan besteht dabei aus allen zuvor generierten Scans. Der neueste Scan wird mit dem gesamten Metascan registriert um Korrespondenzen zu finden (siehe Abb. 4.3). Dadurch, dass der Metascan alle zuvor generierten Scans beinhaltet, können eher richtige Korrespondenzen gefunden werden. Der Aufwand des Metascanmatching nimmt mit jedem neu generierten Scan deutlich zu, deswegen ist es wichtig Algorithmen zur Ausdünnung des Scans einzusetzen. Auch bei dieser Methode akkumulieren sich Fehler, allerdings sind die Auswirkungen im Vergleich zur inkrementellen Registrierung deutlich geringer.

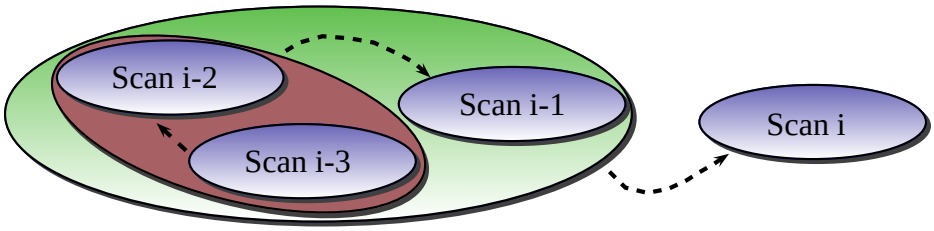


Abbildung 4.3: Metascanmatching

4.4 Datenreduktion

Da die Anzahl der Punkte im Metascan mit zunehmender Anzahl an Scans immer größer wird, ist es nötig diese Punkte wiederum mit einem Minimum an Informationsverlust zu reduzieren.
Eine Möglichkeit, die auch im Rahmen dieser Bachelorarbeit implementiert wurde, sind die *sparse point maps* [HB10]. Die Kernidee hinter diesen ist, dass das Speichern doppelter Punkte vermieden werden soll. Um dies zu erreichen wird eine zusätzliche Korrespondenzensuche durchgeführt und alle Punkte entfernt, die einem Punkt in der realen Umgebung entsprechen, zu dem schon ein Punkt in der *sparse point map* existiert.
Die Korrespondenzen sind hierbei wieder wie beim ICP-Algorithmus definiert: Ein Punkt $\boldsymbol{d} \in D_i$ wird nicht zu M hinzugefügt, wenn die Punkt-zu-Punkt Distanz zum nächstgelegenen Punkt $\boldsymbol{m} \in M$ kleiner als ein Minimum der erlaubten Distanz $d_{\min}$ ist:

$$M_i = M_{i-1} \cup \{\boldsymbol{d}_{i,j} | \nexists \boldsymbol{m}_{i-1,k} \in M_{i-1} : \|\boldsymbol{d}_{i,j} - \boldsymbol{m}_{i-1,k}\| < d_{\min}\} \tag{4.2}$$

Durch $d_{\min}$ wird eine Region aufgespannt, in der sich nur ein Punkt befindet. Damit stellt $d_{\min}$ eine obere Grenze für die Punktdichte in der *sparse point map* M dar. Wird ein großer Wert für $d_{\min}$ gewählt, so werden die aufgespannten Regionen kleiner, d. h. die Punktdichte größer. Umgekehrt werden durch ein kleines $d_{\min}$ die Regionen größer. Dabei ist zu beachten, dass mit entsprechend groß gewählten $d_{\min}$ auch der Detailgrad der Umgebung abnimmt und somit grobkörniger wird.

Kapitel 5

Umsetzung und Ergebnisse

In der ersten Phase der Erstellung dieser Arbeit wurde die schon in Robbie 12 von Peter Schneider in seiner Studienarbeit(2006)[Sch07] integrierte 6D-SLAM Lösung unter Berücksichtigung von Konvertierbarkeit und Effizienz untersucht.
Das Ergebnis dieser Untersuchung war, dass der Aufwand der Konvertierung zu groß sei und abzusehen war, dass die Effizienz ebenfalls nicht gegeben sein wird.
Aus diesem Grund wurde sich auf die Lösung des 6D-SLAM-Problems von Dagmar Lang für das Velodyne-Projekt [PLNP10] beschränkt, welches in das Robbie-Framework zu integrieren galt.
Diese Implementierung bietet eine Schnittstelle zur Anwendung von verschiedenen Bibliotheken der nächsten Nachbar Suche. Daher wurde zunächst evaluiert, welche der beiden Bibliotheken zum Suchen des nächsten Nachbarn eines Punktes für unsere Zwecke, d. h. im dreidimensionalen Raum, ein besseres Ergebnis liefert. Besser heißt hier wiederum schneller.
Zur Evaluation der beiden implementierten Scanmatching Verfahren wurde die Arena so umgebaut, dass sie einfach gehalten und dadurch auf der erzeugten 3D-Karte nachvollziehbar ist. Außerdem muss die Qualität der Verfahren gut erkennbar sein. Im Detail befinden sich, wie auf der Abb. 5.1 zu erkennen, zwei geschlossene Blöcke in der Mitte der Arena mit einem angrenzendem *stepfield*. Außen herum befindet sich der rechteckige 1,20 m breite Pfad den Robbie abfahren kann. Der Pfad ist auf der rechten längeren Seite mit seitlich angeordneten 15° Rampen ausgelegt. Auf der linken Seite befinden sich 4 aufeinander zulaufende 15° Rampen.

Als zusätzliche Anmerkung sei darauf hingewiesen, dass in der getesteten Revision von Robbie 16 die *obstacle detection* deaktiviert ist. Der Grund dafür ist die während der RoboCup German Open 2011 herausgestellte hohe Fehleranfälligkeit. Da Robbie nun keine Hindernisse unterhalb des 2D-Laserscanners erkennen konnte, wurde das *stepfield* (Erkennbar auf Abb. 5.1) so modifiziert, dass einzelne Blöcke vertikal vergrößert wurden, so dass der 2D-Laserscanner diese erkennen kann.

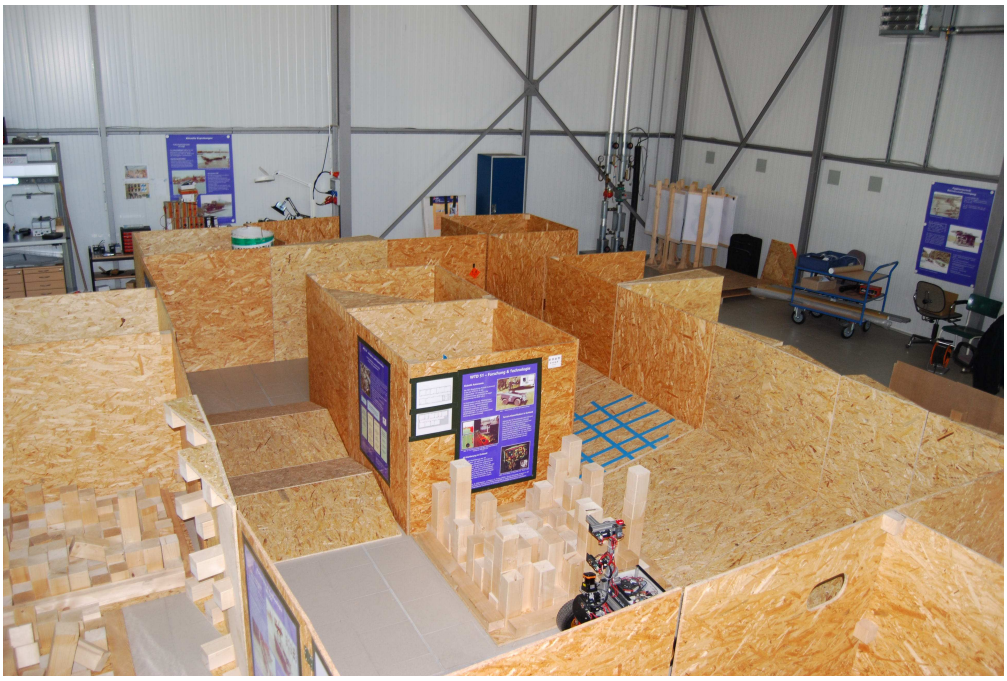


Abbildung 5.1: Versuchsaufbau der Rescue-Arena für die Evaluation der Scan Matching Verfahren

Das System auf dem die Evaluierungen ausgeführt wurden hat folgende Konfiguration:

- Prozessor: Intel Core i7-2630QM
- Grafikkarte: NVIDIA GeForce GT 540M mit 1024 MB GDDR3
- Arbeitsspeicher: 8 GB SO-DIMM DDR3 RAM mit 1333 MHz
- Betriebssystem: Ubuntu 10.10
- Linux Kernel-Version: 2.6.35-28
- Compiler: gcc version 4.4.5

Zunächst wird nun auf die nächste Nachbar Suche, dessen Implementation sowie Evaluation eingegangen. Danach folgen die Analysen der beiden Scanmatching Verfahren der paarweisen Registrierung sowie das Metascanmatching. Da beim Metascanmatching eine Datenreduktion wichtig ist, wird hierbei das implementierte Verfahren der *sparse point maps* [HB10] evaluiert.

5.1 Nächste Nachbar Suche

Wie schon in Abschnitt 4.1 beschrieben, ist eine schnelle nächste Nachbar Suche essentiell für die Lösung des 6D-SLAM in Echtzeitanwendungen wie Robbie. Da es für dieses Problem frei verwendbare Bibliotheken gibt, ist ein Ziel dieser Arbeit die Effizienz, zweier gängiger Bibliotheken zu messen, um die für unsere Anwendung bessere zu wählen.

5.1.1 Implementation der nächsten Nachbar Suche

Im *Slam3DWorker* befindet sich mit dem *NNFinder* eine Schnittstelle zur Anbindung von verschiedenen Bibliotheken zur nächsten Nachbar Suche. Zur Evaluation der nächsten Nachbar Suche wurde eine Testklasse für das Robbie-Framework geschrieben die auf diese Schnittstelle zugreift und mit der ein einfaches einbinden und evaluieren verschiedener Bibliotheken für die nächste Nachbar Suche möglich sein soll.
Die Evaluation selbst erfolgt dabei durch den Vergleich der Laufzeiten der jeweiligen Bibliotheken.
Zur Evaluation wurde die *EvaluateNNS*-Klasse integriert, welche über das Profil *evaluateNNS* aufgerufen wird (siehe Anhang A.3 für die Konfiguratonsmöglichkeiten)

5.1.2 Evaluation der Nächsten Nachbar Suche

Für diese Arbeit wurden die die beiden Bibliotheken ANN in Version 1.1 und FLANN evaluiert, wobei FLANN zum Einen eigenständig mit der Version 1.2 und zum Anderen über OpenCV, mit unbekannter Versionsnummer eingebunden ist. Zusätzlich wurde eine naive Implementation mit quadratischem Aufwand zu Vergleichszwecken getestet.
Evaluiert wird, indem die verschiedenen Suchalgorithmen eine bestimmte Anzahl an nächsten Nachbarn, bei festgelegtem maximalem Abstand, finden müssen. Während der Testläufe hat sich herausgestellt, dass der maximale Abstand zwischen zwei Punkten bei der nächsten Nachbar Suche bedeutend für die Laufzeit ist. Aus diesem Grund wurden Testläufe mit maximalen Abständen von 10 mm und 100 mm durchgeführt.
Zur Feststellung der Laufzeit werden Zeitstempel verwendet. Somit ist die konkrete Laufzeit nicht allgemeingültig, aber das Laufzeitverhältnis der Bibliotheken zueinander sollte im Allgemeinen auf verschiedenen Systemen gleich bleiben.
Da ein 3D-Scan von Robbie etwa 100.000 Punkte beinhaltet, wurden iterativ die Laufzeiten der Suche für eine Punktmenge von 100 bis 100.000 Punkten gemessen.

Dabei wurde als Intervallschritt 100 gewählt, d. h. es wurde erst zu 100 Punkten der nächste Nachbar gesucht, dann zu 200, zu 300, usw.
Wie schon in Abschnitt 4.1.1 und 4.1.2 beschrieben, bieten ANN und FLANN zwei verschiedene Suchmethoden an, somit wurde für ANN und FLANN sowohl die direkte nächste Nachbar Suche als auch die Suche nach nächsten Nachbarn in einem festgelegten Radius untersucht.
Im Vorfeld ist zu erwähnen, dass festgestellt wurde, dass die OpenCV-Integration von FLANN langsamer als die naive nächste Nachbar Suche ist. Wahrscheinlich ist der Grund hierfür eine veraltete OpenCV Version im Robbie-Framework oder ein anderer Softwarefehler. Aus diesem Grund ist die OpenCV Version schon im Vorfeld ausgeschieden.

Bibliothek zur nächsten Nachbar Suche	Initialisierungszeit für Punkteanzahl			
	10.000	50.0000	100.000	200.000
ANN (mit nächster Nachbar Suche)	5 ms	24 ms	48 ms	110 ms
ANN (mit Radius Suche)	5 mm	23 ms	51 ms	106 ms
FLANN (mit nächster Nachbar Suche)	4 ms	18 ms	38 ms	73 ms
FLANN (mit Radius Suche)	4 ms	17 ms	36 ms	75 ms

Tabelle 5.1: Initialisierungszeiten von ANN und FLANN bei einer Anzahl von 10.000, 50.0000, 100.000 sowie 200.000 Punkten

In der Tabelle 5.1 sind für die konkreten Werte 10.000, 50.0000, 100.000 sowie 200.000 die Initialisierungszeiten dargestellt. Man kann anhand der Werte deutlich erkennen, dass die Initialisierungszeiten im Vergleich zu den Laufzeiten vernachlässigbar gering sind. Der mögliche Einfluss eines unterschiedlich großen Suchradius auf die Initialisierungszeit wurde ebenfalls untersucht. Dabei wurde festgestellt das der Suchradius keinen Einfluss auf diese hat.
Es wurde erwartet, dass die Initialisierungszeit keinen unerheblichen Anteil an der Gesamtlaufzeit haben wird, allerdings reicht es für die Suche nach Punkten im dreidimensionalen Raum aus, nur einen einzigen k-d-Baum aufzubauen, was den Initialisierungsaufwand gering hält.
In den Tabellen 5.2 und 5.3 sind für die konkreten Werte 10.000, 20.0000, 60.000 und 100.000 die Laufzeiten bei einem maximalen Abstand von 10 mm bzw. 100 mm dargestellt.
Die besten Ergebnisse lieferte ANN bei der Suche nach dem direkten nächsten Nachbarn. Dabei ist ANN mit direkter nächster Nachbar Suche in jedem Fall schneller als alle anderen Konfigurationen. Sehr interessant ist, dass ANN mit Radius-Suche um ein vielfaches langsamer, und auch langsamer als FLANN mit Nächster Nachbar Suche bzw. Radius-Suche ist.

Bibliothek zur nächsten Nachbar Suche	Laufzeit für die Punkteanzahl			
	10.000	20.0000	60.000	100.000
ANN (mit nächster Nachbar Suche)	19 ms	37 ms	124 ms	326 ms
ANN (mit Radius Suche)	30 ms	54 ms	195 ms	540 ms
FLANN (mit nächster Nachbar Suche)	43 ms	91 ms	319 ms	558 ms
FLANN (mit Radius Suche)	72 ms	138 ms	311 ms	419 ms

Tabelle 5.2: Laufzeiten von ANN und FLANN bei einem maximalem Abstand von 10 mm zum nächsten Nachbarn

Bibliothek zur nächsten Nachbar Suche	Laufzeit für die Punkteanzahl			
	10.000	20.0000	60.000	100.000
ANN (mit nächster Nachbar Suche)	9 ms	19 ms	84 ms	163 ms
ANN (mit Radius Suche)	937 ms	2338 ms	4050 ms	4245 ms
FLANN (mit nächster Nachbar Suche)	42 ms	91 ms	319 ms	557 ms
FLANN (mit Radius Suche)	76 ms	154 ms	428 ms	657 ms

Tabelle 5.3: Laufzeiten von ANN und FLANN bei einem maximalem Abstand von 100 mm zum nächsten Nachbarn

Deutlich wird der Unterschied der Laufzeiten von ANN und FLANN im Diagramm 5.2 für den maximalen Abstand von 10 mm, sowie im Diagramm 5.3 für einen maximalen Abstand von 100 mm sichtbar.

Letztendlich kann mit dem Ergebnis der Evaluation gesagt werden, dass ANN mit der direkten nächsten Nachbar Suche für das Finden korrespondierender Punkte im dreidimensionalen Raum in jedem Fall am Besten geeignet ist. Sollten allerdings mehrere Nachbarn in einem festgelegten Radius um einen Punkt gesucht werden, so ist FLANN mit Radius Suche signifikant schneller als ANN.

5.2 Paarweise Registrierung

Die Paarweise Registrierung wurde in Abschnitt 4.3.1 erklärt. In diesem Abschnitt soll nun auf die Implementation dieses Algorithmus in das Robbie-Framework eingegangen werden, worauf die Evaluation der erzeugten 3D-Karten mit Hilfe dieser Implementierung folgt.

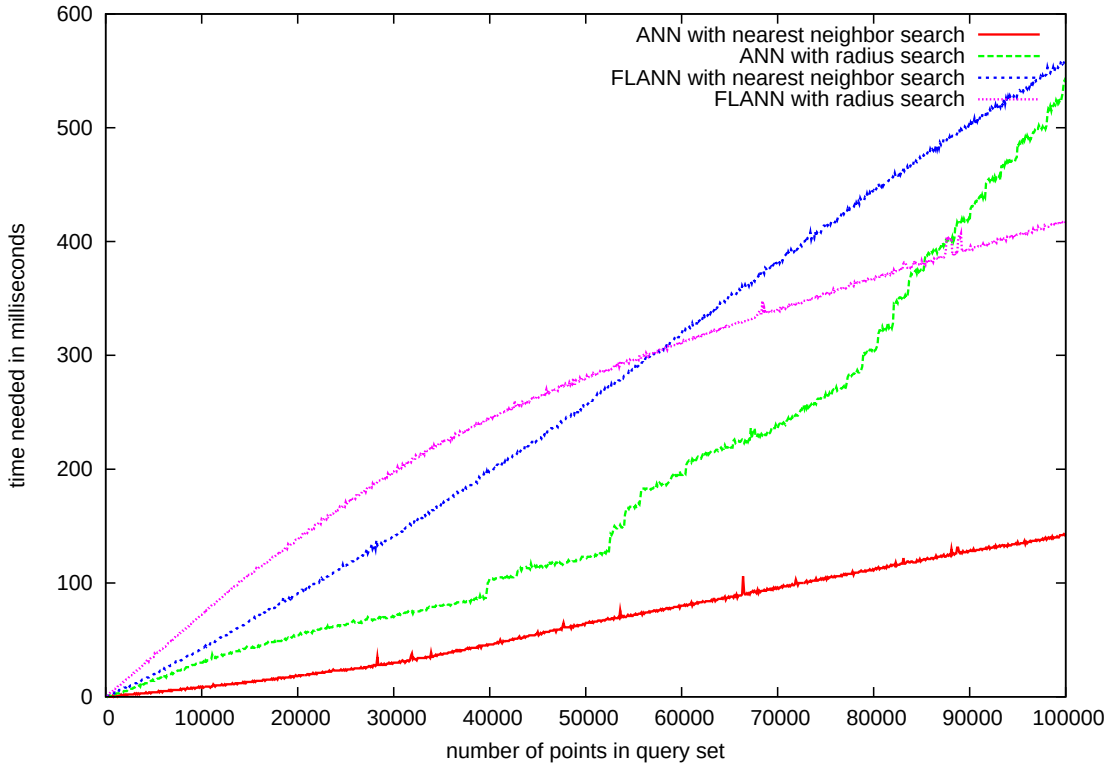


Abbildung 5.2: Laufzeiten von ANN und FLANN bei einem maximalem Abstand von 10 mm zum nächsten Nachbarn

5.2.1 Implementation des ICP-Algorithmus bzw. der paarweisen Registrierung

Die Implementierung des ICP-Algorithmus, des Graphen zur Speicherung der 3D-Scans, sowie die paarweise Registrierung der 3D-Scans wurde von dem Velodyne-Projekt [PLNP10] übernommen und an das Robbie-Framework angepasst.
Für die Lösung des SLAM-Problems sind dabei zwei *Worker* zuständig.

Der Slam3DWorker bietet eine Schnittstelle zu den Bibliotheken der Berechnung der nächsten Nachbarn von *Vector3D*-Punkten. Weiterhin ermöglicht er die Berechnung des ICP-Algorithmus und speichert die Punkte der Punktwolke in Form der Klasse *Scan*.

Die *Scan* Klasse entspricht dabei einem gewöhnlichem Container, wie etwa einem *vector*. Der Unterschied besteht darin, dass diese nicht generisch ist und nur auf *Vector3D-Punkte* der Eigen2-Bibliothek[1] festgelegt ist und zu-

[1] eigen2

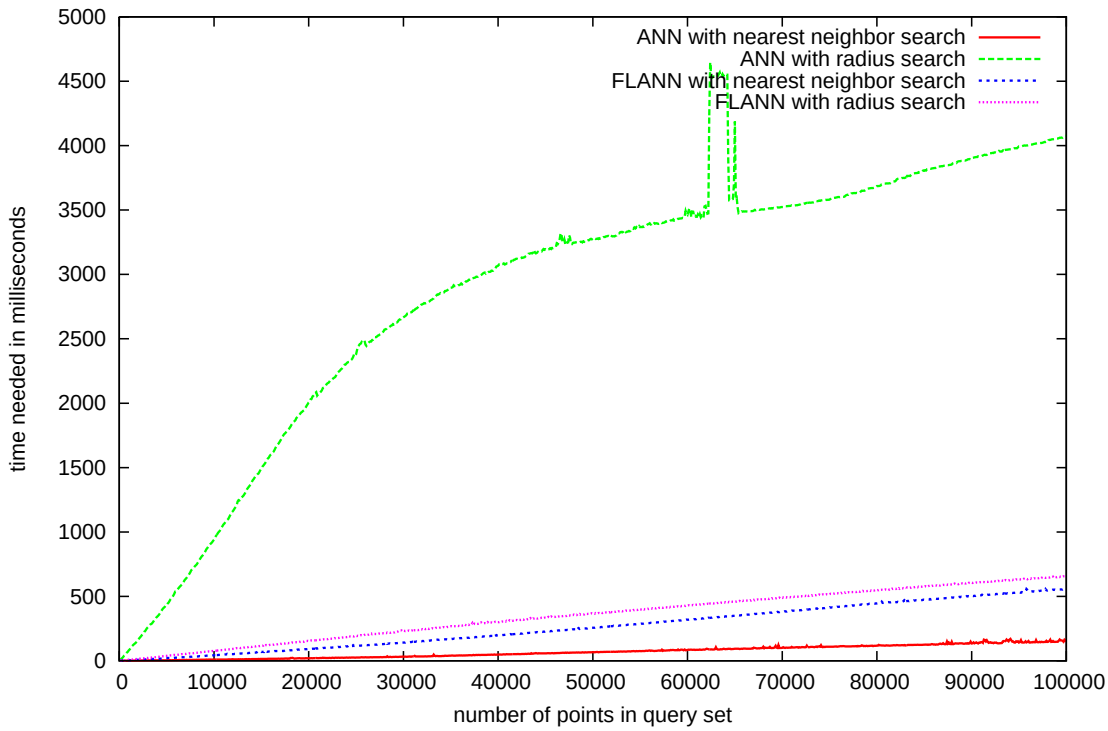


Abbildung 5.3: Laufzeiten von ANN und FLANN mit der Suche nach den nächsten Nachbarn mit maximalem Abstand von 100 mm

sätzlich alle beinhaltenden Punkte um ein übergebenes $\boldsymbol{R}$ und $\boldsymbol{t}$ rotieren und translatieren lassen kann.

Im Laufe dieser Bachelorarbeit wurde der *Worker* um die *PointReductionAlgorithms*-Klasse zur Reduzierung der Anzahl der Punkte im Scan erweitert. Da diese Klasse nur Algorithmen anbieten und keine Daten speichern soll, verfügt sie über einen privaten Konstruktor. Die Algorithmen selbst werden als statische Funktionen angeboten.

Der SLAMGraph Worker beinhaltet die Datenstruktur zur Speicherung der verschiedenen Scans. Die Datenstruktur entspricht dabei einer Baumstruktur mit Knoten, die über Kanten miteinander verbunden sind. In den Knoten sind dabei die einzelnen Scans gespeichert. Jeder Knoten kann mit beliebig vielen anderen Knoten verbunden sein. Dabei werden verschiedene Typen von Eingangs- sowie Ausgangskanten gespeichert. In den Kanten selbst ist die Rotationsmatrix $\boldsymbol{R}$ sowie der Translationsvektor $\boldsymbol{t}$ gespeichert.

Über den Parameter *iScanMatchType* in der *Rescue.xml* kann dann die gewünschte Transformation ausgewählt werden.

Durch die Speicherung der doppelten Kanten ist es einfach möglich, einen Scan A in das Koordinatensystem eines Scans B zu transformieren, indem alle Punkte aus A mit der Rotationsmatrix und dem Translationsvektor der Ausgangskante zu B transformiert werden. Aber genauso einfach ist es auch möglich B in das Koordinatensystem von A zu transformieren, indem die Punkte aus B mit der Rotationsmatrix und dem Translationsvektor der Eingangskante von A nach B transformiert werden.

Zur Traversierung des Baumes wird der bekannte Dijkstra-Algorithmus [Dij59] verwendet.

Aufgerufen werden die *Worker* über das *Slam3DModule*, welches die Laserscan-Daten empfängt, diese im 3D-Scan speichert und den 3D-Scan an den *SLAMGraph* übergibt.
Außerdem ist es über das *Slam3DDumpModule* möglich die 3D-Scans sowie die Rotationsmatrizen und Translationsvektoren in Dateien abzuspeichern um sie mit dem *Slam3DLoadDumpModule* aufzurufen. Der Vorteil davon ist, dass immer auf den gleichen Datensätzen getestet werden kann und dies weitgehend unabhängig von anderen Berechnungen des Robbie-Frameworks.

Zur Ermittlung der xy-Position sowie des θ_y-Winkels sollten die vom *SlamModule* errechneten Informationen verwendet werden. Der θ_x und θ_z-Winkel kann über die IMU ermittelt werden. Nur die z-Position lässt sich über keinen Sensor direkt ermitteln, daher soll z als 0 angenommen werden.
Somit können alle 6 Freiheitsgrade ermittelt und daraus die richtige Poseänderung im dreidimensionalen Raum abgeleitet werden. Allerdings hat sich dieses Vorgehen praktisch nicht bewährt. Die 3D-Scans wurden stets falsch um θ_y rotiert. Der Grund hierfür liegt wahrscheinlich in der Berechnung der Rotation im *SlamModule* zur Erstellung der 2D-Karte. Zur Ermittlung der Rotation wird dort der in Kapitel 3.1 erwähnte Partikelfilter verwendet. Da die Datendichte für die Lösung des dreidimensionalen SLAM-Problems deutlich geringer als im zweidimensionalen Fall ist, gibt es weniger Partikel. Dadurch ist die Wahrscheinlichkeit größer eine falsche Poseinformation zu erhalten.
Aus diesem Grund wurde nachfolgend auf die Poseinformationen des Szenengraphen von Robbie gesetzt. Wie in Abschnitt 2.5.1 erwähnt, kann aus diesem ebenfalls die Pose-Informationen ermittelt werden. Aus diesen Daten konnten nun die Scanmatching-Vefahren der paarweisen Registrierung sowie das Metascanmatching evaluiert werden.

5.2.2 Evaluation

Die Evaluation der paarweisen Registrierung erfolgte durch visuelle Beurteilung. Dabei wurde zuerst die durch reine Odometriedaten erstellte 3D-Karte (siehe Abb. 5.4 herangezogen. Anhand dieser 3D-Karte sind durch Odometriefehler entstandene, falsche Transformationen gut erkennbar.

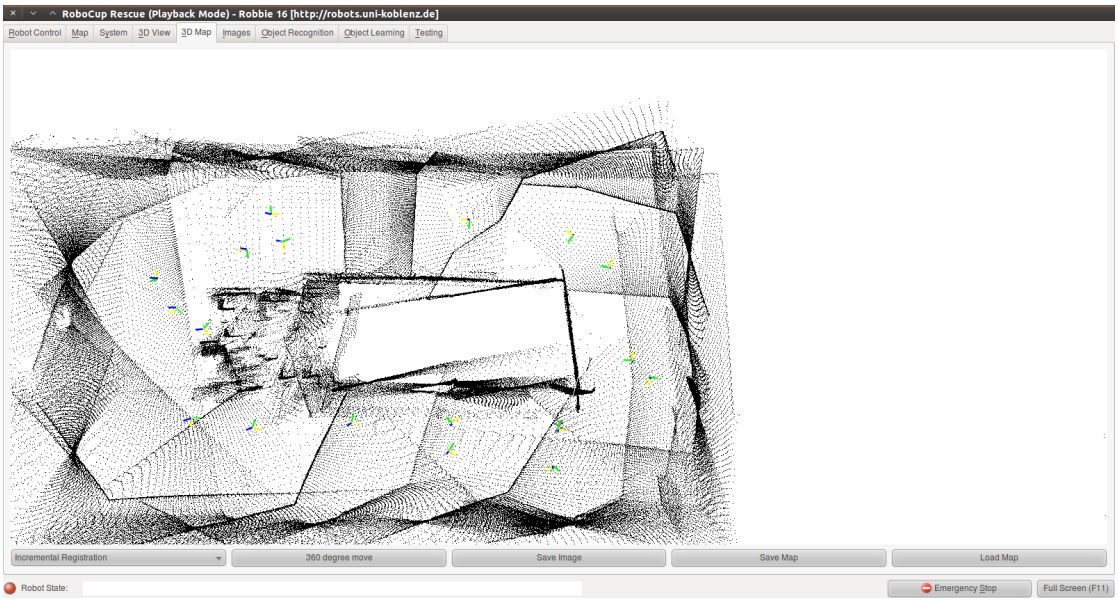


Abbildung 5.4: Die über reine Odometriedaten erstellte 3D-Karte der Arena. Anhand des mittleren Blocks, sowie dem äußeren rechten Rand, können Fehler der 3D-Karte gut erkannt werden.

Daraufhin wurde mit verschiedenen Parametereinstellungen die 3D-Karte durch paarweise Registrierung der 3D-Scans erstellt. Als Parameter können hier der maximale Abstand korrespondierender Punkte über *fMaxDist* sowie die Anzahl der maximalen Iterationsschritte des ICP-Algorithmus über *iErrorThreshold* gewählt werden. Folgende Ergebnisse brachte die Evaluation:
Bei der Erstellung der 3D-Karte mit Hilfe der paarweisen Registrierung werden einzelne 3D-Scans, wie in Abb. zu erkennen ist, falsch transformiert. Aufgrund dieser falsch eingetragenen 3D-Scans leidet die Qualität der 3D-Karte erheblich, da, wie in Abschnitt 4.3.1 erwähnt, sich Fehler stark akkumulieren.
Das Ergebnis für die paarweise Registrierung ist fatal. Allerdings kann dies nicht am Algorithmus selbst liegen. Wahrscheinlich liegt die Ursache an einem Softwarefehler der bis zum Abschluss dieser Arbeit nicht gefunden werden konnte. Es ist zumindest auszuschließen, dass der ICP-Algorithmus fehlerhaft arbeitet, da dieser beim nachfolgenden Metascanmatching gute Ergebnisse liefert.

Maximaler Abstand korrespondierender Punkte in mm	Maximale Iterationstiefe des ICP-Algorithmus	visuelle Beurteilung der Karte
100	25	nicht ok
100	50	nicht ok
125	25	nicht ok
125	50	nicht ok
150	25	nicht ok
150	50	nicht ok

Tabelle 5.4: Getestete Werte für die paarweise Registrierung. Dabei wurde bei keinem Wert ein zufriedenstellendes Ergebnis geliefert.

5.3 Metascanmatching

Wie in Abschnitt 4.3.2 beschrieben, entspricht der Metascan einer Akquirierung aller bisher aufgenommen 3D-Scans. In diesem Abschnitt wird auf die konkrete Implementierung in das Robbie-Framework eingegangen und die damit errechneten 3D-Karten evaluiert. Weiterhin wurde zur Reduktion der Daten des Metascans das Verfahren der *sparse point maps* implementiert welches hier ebenfalls evaluiert werden soll.

5.3.1 Implementation des Metascanmatching

Die Implementation des Metascans ist relativ einfach. Wenn der erste 3D-Scan erstellt wurde, wird dieser zum Metascan. Beim nächsten 3D-Scan wird der Metascan mit diesem registriert und der Metascan vergrößert.
Gespeichert wird der Metascan im *Slam3DModule*. Im *SLAMGraph* wird der Metascan aufgebaut und wie schon bei der paarweisen Registrierung wird über den Parameter *iScanMatchType* gesteuert, mit welcher Transformation die erzeugten 3D-Scans registriert werden sollen.

5.3.2 Implementation der Datenreduktion

Zur Datenreduktion des Metascanmatching wurde im Rahmen dieser Bachelorarbeit das Verfahren der *sparse point maps* [HB10] implementiert (siehe Kapitel 4.4). Dabei wird über alle Punkte iteriert und die nächsten Nachbarn im Radius r um den aktuellen Punkt des Metascans $\boldsymbol{P}_i, 0 < i <= n_M$ mit Hilfe der ANN Bibliothek gesucht, wobei M die Punktmenge des Metascans sei und n_M die Anzahl der Punkte von M. Der Radius r wird über den Parameter *minDist* festgelegt.

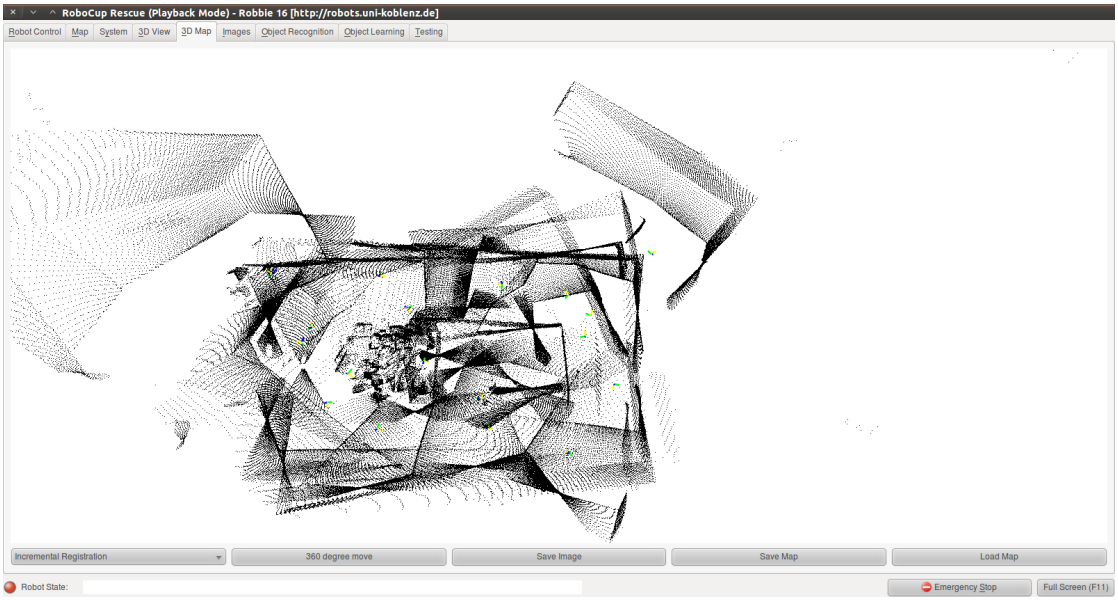


Abbildung 5.5: Die über paarweise Registrierung erstellte 3D-Karte der Arena. Die 3D-Karte beinhaltet einige falsch transformierte Scans und ist damit unbrauchbar.

Die von ANN gelieferten nächsten Nachbarn müssen nochmals im Metascan gesucht werden, um aus diesem entfernt zu werden. Daraus ergibt sich wiederum eine zweite Iteration über M. Über den Parameter *maxRemovedNeighbors* ist außerdem eine obere Grenze zum Löschen von Nachbarn festgelegt.

Während der Iteration sollte n_M zunehmend abnehmen. Außerdem müssen für jeden Punkt $\boldsymbol{P}_i$ die nächsten Nachbarn gefunden werden. Problem dabei ist, dass ein Aufbauen des k-d-Baums für jeden Iterationsschritt sehr ineffizient ist, und somit die nächsten Nachbarn im ursprünglich aufgebauten Baum gesucht werden müssen, obwohl dieser noch Punkte enthält, die schon entfernt wurden.

Deutlich effizienter wäre es über den k-d-Baum der Bibliothek zur nächsten Nachbar Suche, allerdings wird ein direkter Zugriff auf diesen k-d-Baum nicht ermöglicht.

5.3.3 Evaluation

Die in Abb. 5.6 erkennbare, per Metascanmatching erstellte 3D-Karte, ist im Vergleich zur reinen Odometrie basierten 3D-Karte aus Abb. 5.4 deutlich besser. Erkennbar wird dies anhand des mittleren Blocks, sowie dem äußeren rechten Rand. Dennoch konnten nicht alle Odometriefehler ausgeglichen werden. Dies ist am rechten Rand des mittleren Blocks gut erkennbar.

Zur Erstellung dieser 3D-Karte wurde die maximale Distanz korrespondierender Punkte auf 125 mm und die maximalen Iterationsschrite auf 75 festgelegt. Visuell hat diese Parameterwahl das beste Ergebnis erzielt. Dabei wurden die folgenden Parameterkonfigurationen getestet:

Maximaler Abstand korrespondierender Punkte in mm	Maximale Iterationstiefe des ICP-Algorithmus	visuelle Beurteilung der Karte
50	50	nicht ok
100	25	ok
100	40	ok
100	50	nicht ok
125	20	nicht ok
125	25	nicht ok
125	50	ok
125	75	am besten
150	25	ok
150	50	nicht ok
500	200	nicht ok

Tabelle 5.5: Getestete Werte für das Metascanmatching.

Da die paarweise Registrierung leider keine brauchbaren Ergebnisse liefert ist ein direkter Vergleich der beiden Scanmatching Methoden nicht möglich.

Weiterhin wurde die implementierte *sparse point map* evaluiert. Wie in der Abbildung 5.8 im Vergleich zur Abbildung 5.7 zu erkennen, liefert die *sparse point map* ein sehr gutes Ergebnis. Trotz Punktreduktion bleiben Strukturen der Umgebung erhalten. Die Minimaldistanz zwischen zwei Nachbarpunkten wurde hier mit 10 mm und die maximale Anzahl an Nachbarn, die in dem gewählten Radius um einen Punkt entfernt werden, mit 25 gewählt.
Trotz des sehr guten Ergebnisses ist die verwendete Implementation der *sparse point map* auf Grund des hohen Berechnungsaufwands nicht für eine Echtzeitanwendung wie Robbie geeignet.

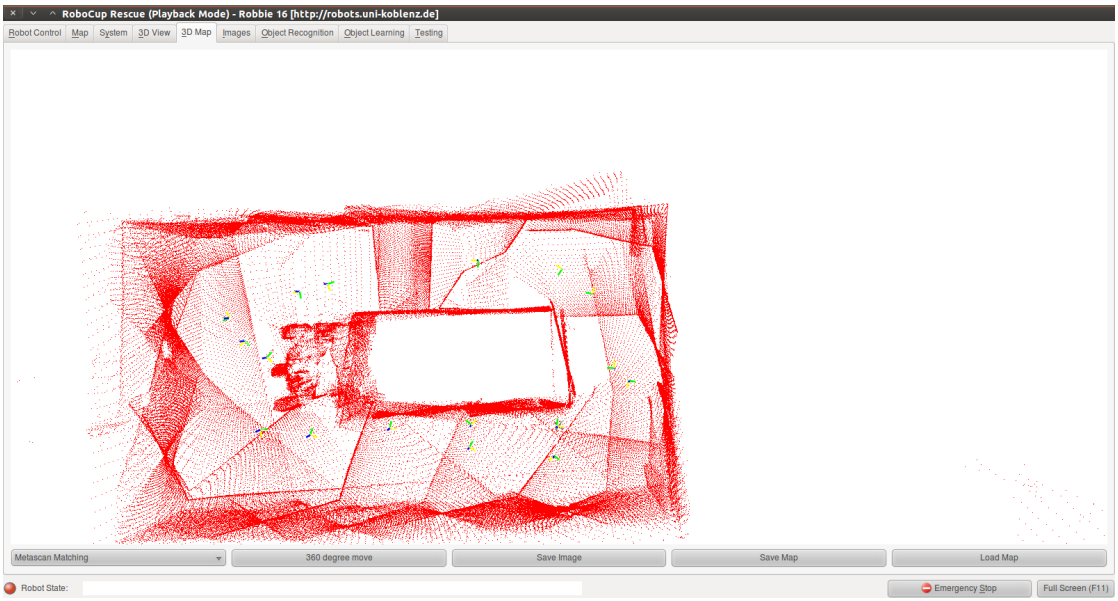


Abbildung 5.6: Draufsicht auf die durch Metascanmatching erstellte 3D-Karte der Arena. Die Qualität der Karte ist gut. Lediglich auf der rechten Seite der 3D-Karte liegen die Wände der Arena nicht übereinander

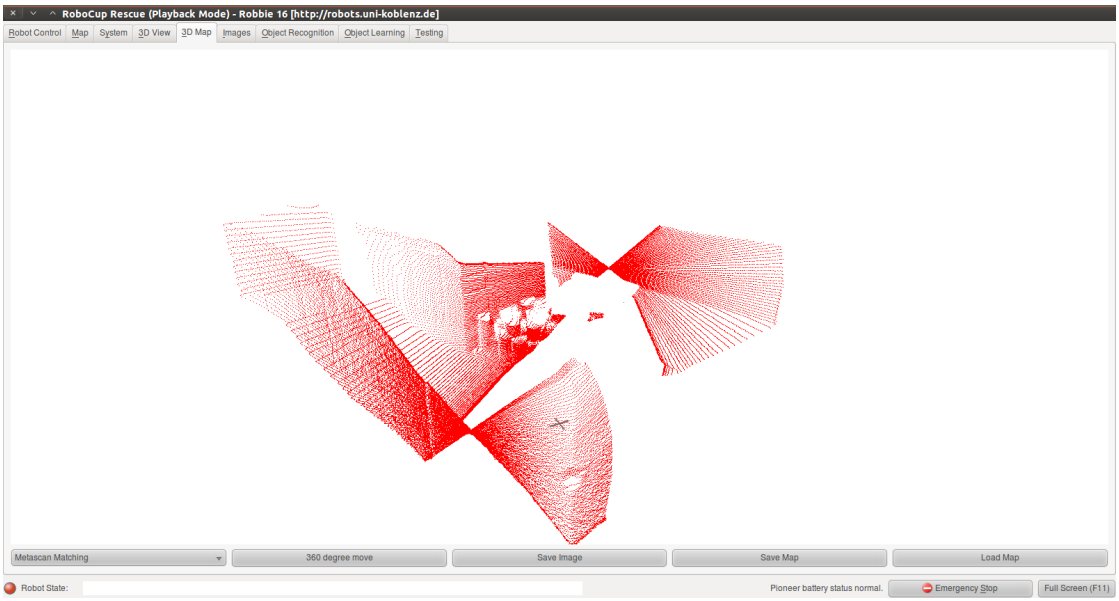


Abbildung 5.7: 3D-Scan ohne Anwendung der *sparse point map*

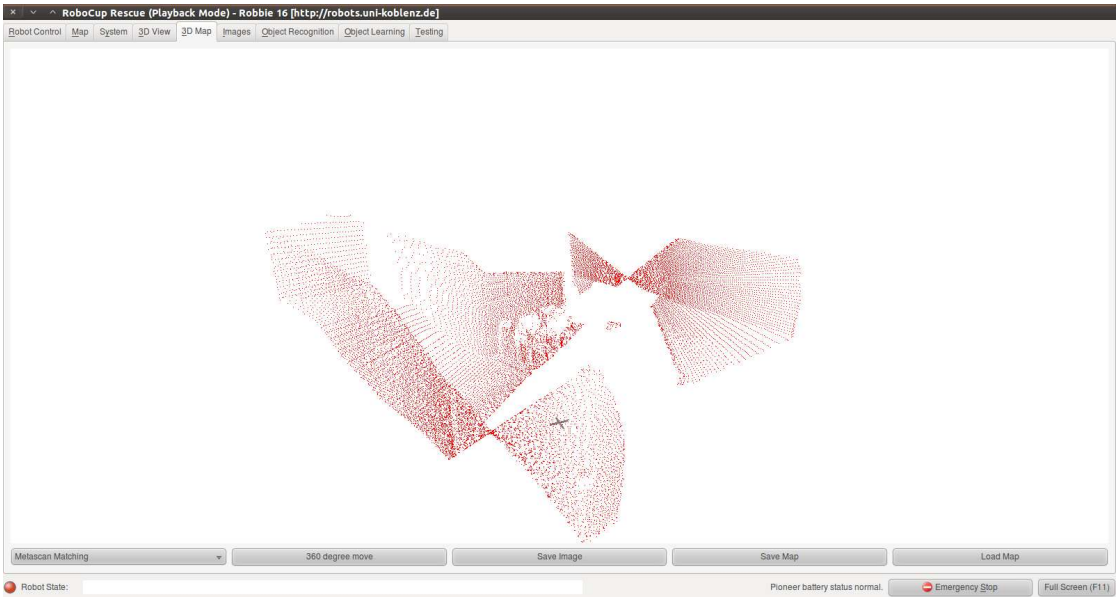


Abbildung 5.8: 3D-Scan mit Anwendung der *sparse point map*

Kapitel 6

Zusammenfassung und Ausblick

Rettungsroboter sollen in der Zukunft eingesetzt werden, um in Gebieten Hilfe zu leisten, die für Menschen zu gefährlich sind. Um die Bedienung zu vereinfachen und weil eine ständige Verbindung zum Roboter möglicherweise nicht immer garantiert werden kann, sollten sich diese Roboter autonom in ihrer Umgebung fortbewegen können. Für dieses autonome Vorgehen benötigen Roboter Karten. Die vorliegende Arbeit versuchte den Schritt von 2D- auf 3D-Karten zu gehen und mit diesen ein möglichst genaues Bild der Umgebung zu erstellen.

Anfangs wurden dabei die Ursache für die Gründung der RoboCup Rescue Liga und deren Ziele genannt und es wurde Robbie, seine Sensoren mit denen er seine Umwelt wahrnimmt, sowie der Aufbau seiner Softwarearchitektur beschrieben. Ein Augenmerk wurde hier natürlich auf den Laserscanner und die erzeugten 3D-Scans gelegt, aus denen die 3D-Karte errechnet werden soll. Das Errechnen der Karten stellt eines der größten Probleme in der autonomen Robotik dar und ist als das SLAM-Problem bekannt.

Für dieses Problem gibt es zwei grundlegende Verfahrensansätze: probabilistisch und Graph-basiert. Einige Verfahren die Pellenz [Pel10] darlegte, wurden kurz veranschaulicht. Außerdem wurden von Holz und Behnke in [HB10] vorgestellte Verfahren zur Reduktion von Punkten in Scans aufgezeigt und das Vorgehen anderer RoboCup Rescue Teams exemplarisch, an dem Team CASualty und Team Pelican United, veranschaulicht.

Ein wichtiger Teil dieser Arbeit ist die Kartierung. Diese kann in drei Schichten unterteilt werden: Auf der untersten Ebene steht die nächste Nachbar Suche. Diese muss besonders schnell sein, weshalb die zwei Bibliotheken ANN und FLANN analysiert wurden, die dieses Problem über Bäume lösen. Die Evaluation ergab, dass ANN mit der direkten Suche nach Nächsten Nachbarn im dreidimensionalen Raum deutlich besser geeignet ist.

Ausgeführt wird die nächste Nachbar Suche vom ICP-Algorithmus, der zweiten Schicht der Kartierung. Hier ist besonders die Kostenfunktion, die es zu minimieren gilt, von Bedeutung. Die Minimierung findet dabei iterativ statt. Während der Evaluation der Scanmatching Verfahren stellte sich heraus, dass der ICP-Algorithmus nicht geeignet ist für eher selten eintreffende 3D-Scans, wie dies bei Robbie der Fall ist.
Die oberste Schicht ist das Scanmatching, was die Art des Zusammenfügens verschiedener Scans beschreibt. Dabei wurden zwei grundlegende Arten unterschieden: Die paarweise Registrierung registriert den aktuellen Scan mit dem vorherigen. Dabei werden zwei Scans betrachtet, was sehr effizient ist. Allerdings akkumulieren sich die Fehler hier besonders stark. Um die Auswirkungen dieser Fehler abzuschwächen ist dieser Algorithmus erweiterbar, so dass jeweils solange der aktuelle Scan mit dem vorherigen registriert wird, bis keine signifikante Änderung der Translation mehr stattfindet.
Anders wird beim Metascanmatching vorgegangen. Hier wird der neue Scan mit allen zuvor in einem Metascan vereinigten Scans registriert. Dadurch ist es eher möglich richtige korrespondierende Punkte zwischen den Scans zu finden. Allerdings ist dieses Verfahren sehr rechenaufwendig, da der Metascan mit jedem neuen Scan anwächst. Deswegen wurde das Verfahren der *sparse point maps* untersucht, um die Punkte des Metascans zu reduzieren, aber gleichzeitig die Strukturen der Umgebung so gut es geht zu erhalten.
Das Verfahren der *sparse point maps* wurde implementiert und die Ergebnisse sind sehr zufriedenstellend. Allerdings ist die verwendete Implementierung sehr rechenaufwendig und deswegen leider nicht für Echtzeitanwendungen geeignet. Das Metascanmatching selbst wurde ebenfalls implementiert, jedoch konnten aufgrund eines unbekannten Sofwarefehlers keine sinnvollen Ergebnisse mit diesem Verfahren erzielt werden.

Für die weitere Entwicklung der 3D-Kartierung sollte vor allem der ICP-Algorithmus möglichst beschleunigt werden. Dieser lieferte beim Metascanmatching gute Ergebnisse.
Es sollte aber auch überprüft werden, inwiefern andere Algorithmus zum Finden korrespondierender Nachbarpunkte wie z. B. der *iterative matching range points* Algorithmus [LM94] oder *iterative dual correspondences* Algorithmus [LM94] geeignet sind. .
Da der ICP-Algorithmus den aufwendigsten Berechnungsschritt der Scanmatching Verfahren darstellt bietet sich eine Implementation mit Hilfe von *multi threading* unterstützer nächster Nachbar Suche an, um diesen zu beschleunigen.

Der Metascan hat sich als sehr vielversprechend herausgestellt. Dessen Verwendbarkeit sollte weiter untersucht werden. Dabei sollte dieser soweit wie möglich

„ausgedünnt“ werden um die Berechnungszeit des ICP-Algorithmus zu verkürzen. Aus diesem Grund sollten auch die in Abschnitt 4.4 vorgestellten Datenreduktionsverfahren getestet werden. Gerade die *sparse point map* liefert sehr gute Ergebnisse und für dieses Verfahren könnte eine effizientere Implementation entwickelt werden.
Wenn es keine oder nur wenige korrespondierende Punkte zwischen dem Metascan und dem neuesten 3D-Scan gibt, sollte überprüft werden, inwiefern dieser verwendet werden kann, da dieser 3D-Scan nur von der Odometriemessung abhängig, und damit potenziell eher falsch ist.
Ein Problem stellen falsch eingetragene 3D-Scans im Metascan dar, da diese nur schwer korrigiert werden können. Das gleiche Problem betrifft dynamische Objekte. Hier sollten Methoden entwickelt werden um dem Metascan eine gewisse Dynamik zu ermöglichen.

Anhang A

Konfigurationsanleitung

A.1 Installation

Zur Installation wird Zugriff auf das Robbie 16 Subversion unter `https://svn.uni-koblenz.de/agas/projects/robbie/16` benötigt.
Zur Ausführung der Arbeit müssen folgende Schritte ausgeführt werden:

1. Die Installationsroutine *setup.sh* unter *robbie/16/60_tools/robbie-seu-setup/* ausführen. Dabei werden alle benötigten Bibliotheken installiert und Systemvariablen gesetzt.

2. Im Ordner *robbie/16/30_prog/trunk/* den Unterordner *build* erstellen.

3. Im *build* Ordner *cmake ..* ausführen.

4. Im selben Ordner *make* ausführen. Bei Vorhandensein einer Multi-Core-CPU kann mit dem Zusatz *-jX* der Vorgang beschleunigt werden. *X* steht dabei für die Zahl der Threads zum Kompilieren des Programms.

5. Im selben Programm kann nun Robbie mit *./Robbie* ausgeführt werden. Dafür stehen verschiedene Profile zur Auswahl. Die für die 3D-Kartierung relevanten werden nachfolgend erklärt.

A.2 Relevante Profile für die 3D-Kartierung

Für die 3D-Kartierung stehen verschiedene Profile zur Auswahl die mit *./Robbie PROFILNAME* aufgerufen werden:

- **mapping3DServer** wird von Robbie aufgerufen um die 3D-Kartierung auszuführen.

- **mapping3DClient** wird vom mit dem Server verbundenen Client aufgerufen. Die Kartierung kann sich über den *Map3D*-Tab der GUI angesehen werden.
- **mapping3dServerAlone** wird aufgerufen, wenn auf dem Server selbst die Darstellung der 3D-Karte in der GUI stattfinden soll.
- **mapping3DPlayback** kann aufgerufen werden um die 3D-Kartierung auf vorhandenen *log-files* auszuführen.
- **mapping3DDump** speichert alle 3D-Scans und die Transformationsänderungen in Dateien. Der Ordnerpfad wird in der Konfigurationsdatei festgelegt.
- **mapping3DDumpPlayback** kann die zuvor abgespeicherten 3D-Scans und Transformationsänderungen abspielen.

A.3 Profil zur Evaluierung der nächsten Nachbar Suche

Das Profil *evaluateNNS* befindet sich in der *Testing.xml* und dient der Evaluierung der nächsten Nachbar Suche. Die Evaluierung der nächsten Nachbar Suche lässt sich über über verschiedene Parameter konfigurieren:

- **sDataFile1** gibt den Pfad zur ersten einzulesenden Datei an, die Punkte eines 3D-Scans enthält
- **sDataFile2** gibt den Pfad zur zweiten einzulesenden Datei an, die Punkte eines 3D-Scans enthält
- **sLogFile** gibt den Pfad zur Datei an, die angelegt werden soll um die Ergebnisse zu speichern.
- **iNumPointsInDataFile1** gibt die Anzahl einzulesender Punkte aus *sDataFile1* an
- **iAlgorithm** gibt den Algorithmus an der getestet werden soll. Dabei steht 0 für die naive Suche, 1 für ANN, 2 für FLANN und 3 für die in OpenCV integrierte FLANN Version
- **iSamplingRate** gibt die Rate an in der von *iLowerBound* bis *iUpperBound* nach nächsten Nachbarn gesucht werden soll

- **iLowerBound** gibt die untere Grenze an Suchvorgängen an.
- **iUpperBound** gibt die obere Grenze an Suchvorgängen an.
- **fSearchRadius** gibt den Suchradius in mm an, innerhalb dem sich nächste Nachbar Punkte befinden müssen
- **iSearchMode** gibt den Suchmodus an. Dabei bedeutet 0 die direkte nächste Nachbar Suche und 1 die Radius-Suche

Literaturverzeichnis

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

[BM92] Besl, P. ; McKay, N.: A method for Registration of 3–D Shapes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992, S. 14(2):239 – 256

[Dij59] Dijkstra, E. W.: A note on two problems in connexion with graphs. In: *Numerische Mathematik* Bd. 1. 1959, S. 269 – 271

[EFF98] Eggert, D. W. ; Fitzgibbon, A. W. ; Fisher, R. B.: Simultaneous Registration of Multiple Range Views Satisfying Global Consistency Constraints for Use In Reverse Engineering, 1998

[HB10] Holz, D. ; Behnke, S.: Sancta Simplicitas – On the efficiency and achievable results of SLAM using ICP-Based Incremental Registration. In: *In the Proceedings of International Conference on Robotics and Automation*, 2010

[HHN88] Horn, B. K. P. ; Hilden, H.M. ; Negahdaripour, Sh.: Closed-form solution of absolute orientation using orthonormal matrices. In: *Journal of the Optical Society of America A* (1988), S. 5(7):1127 – 1135

[Hor87] Horn, B. K. P.: Closed-form solution of absolute orientation using unit quaternions. In: *Journal of the Optical Society of America A* (1987), S. 4(4):629 – 642

[LHP+11] Lang, D. ; Häselich, M. ; Prinzen, M. ; Bauschke, S. ; Gemmel, A. ; Giesen, J. ; Hahn, R. ; Haraké, L. ; Reimche, P. ; Sonnen, G. ; Steimker, M. von ; Thierfelder, S. ; Paulus, D.n: RoboCup Rescue – Robot League Team resko@UniKoblenz (Germany) / Universität Koblenz-Landau. 2011. – Forschungsbericht

[LM94] Lu, F. ; Milios, E.: Robot pose estimation in unknown environments by matching 2d range scans. In: *IEEE Computer Vision and Pattern Recognition, CVPR* (1994), S. 249 – 275

[MMS+10] Milstein, A. ; McGill, M. ; Sammut, C. ; Salleh, R. ; Farid, R. ; Miro, J.Valls ; Dissanayake, G. ; Norouzi, M.: RoboCupRescue 2010 - Robot League Team Team CASualty (Australia) / University of New South Wales and University of Technology, Sydney. 2010. – Forschungsbericht

[Mon03] Montemerlo, M.: *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Diss., 2003

[Mou10] Mount, D. M.: *ANN Programming Manual*, 2010. – `http://www.cs.umd.edu/~mount/ANN/Files/1.1.2/ANNmanual_1.1.pdf` (Zugriff am 20.05.2011)

[Muj09] Muja, M.: *FLANN, Fast Library for Approximate Nearest Neighbors*. 2009. – `http://mloss.org/software/view/143/` (Zugriff am 20.05.2011)

[Mül11] Müller, S.: *Vorlesung Computergrafik 2.* `http://userpages.uni-koblenz.de/~cg/ss11/cg2/01_hierachien.pdf`, 2011. – Vorlesungsfolien (Zugriff am 20.05.2011)

[Nüc06] Nüchter, A.: *Semantische dreidimensionale Karten für autonome mobile Roboter*, Rheinische Friedrich-Wilhelms-Universität Bonn, Diss., 2006

[OY09] Ohno, K. ; Yoshida, T.: RoboCupRescue 2009 - Robot League Team <Pelican United(JAPAN)> / Tohoku University and Chiba Institute of Technology. 2009. – Forschungsbericht

[Pel10] Pellenz, J.: *Aktive Sensorik für autonome mobile Systeme*, Universität Koblenz-Landau, Diss., 2010

[PLNP10] Pellenz, J. ; Lang, D. ; Neuhaus, F. ; Paulus, D.: Real-time 3D Mapping of Rough Terrain: A Field Report from Disaster City, 2010

[Sch07] Schneider, P.: *Implementierung von 6D SLAM auf Basis eines schnellen ICP-Algorithmus*, Universität Koblenz-Landau, Studienarbeit, 2007

[SH96] Stoddart, A. ; Hilton, A.: Registration of multiple point sets. In Proceedings of the International Conference on Pattern Recognition, 1996

[TSPP11] Thierfelder, S. ; Seib, V. ; Pellenz, J. ; Paulus, D.: Robot Operating System "Robbie". In: *In Proceedings of WS4C 2011: Workshop on Software Language Engineering for Cyber-physical Systems*, 2011

[web11a] *RoboCup German Open:"Roboter werden überschätzt".* `http://www.heise.de/ct/artikel/RoboCup-German-Open-Roboter-werden-ueberschaetzt-1220464.html`, 5 2011. – Interview der c't mit Adam Jacoff am 01.04.2011 (Zugriff am 20.05.2011)

[web11b] *Robot League History.* `http://www.robocuprescue.org/`, 5 2011. – Offizielle RoboCup Rescue Homepage (Zugriff am 20.05.2011)

[web11c] *Robotics.* `http://uni-koblenz-landau.de/koblenz/fb4/institute/icv/agpaulus/agas-projects/robbie`, 5 2011. – Beschreibung des Robbie Projekts auf der Universitätsseite

[WSV91] Walker, M. W. ; Shao, L. ; Volz, R. A.: Estimating 3-D location parameters using dual number quaternions, 1991, S. 54(10):358 – 367