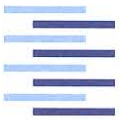


Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Mehmet Bulut

Analyse und Optimierung von Algorithmen zum Lokalisieren mobiler Systeme mit Hilfe von Laserscannern

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Mehmet Bulut

Analyse und Optimierung von Algorithmen zum Lokalisieren mobiler Systeme mit Hilfe von Laserscannern

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Reinhard Baran
Zweitgutachter : Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 9. Mai 2011

Mehmet Bulut

Thema der Bachelorarbeit

Analyse und Optimierung von Algorithmen zum Lokalisieren mobiler Systeme mit Hilfe von Laserscannern

Stichworte

Lokalisierung, Positionsbestimmung, Algorithmen, Optimierung, indoor, Laserscanner, autonom, Fahrzeug, Roboter, mobiles, System, Polygon, FAUST, HAW, Hochschule für Angewandte Wissenschaften, Hamburg

Kurzzusammenfassung

Eine wichtige Teilaufgabe in der Entwicklung eines autonomen Roboters ist die Bestimmung der eigenen Position. Abhängig von den Bedingungen im Einsatzort, der verfügbaren Sensorik und den mechanischen Eigenschaften der mobilen Plattform kommen verschiedene Verfahren zur Positionsbestimmung in Frage. Diese Arbeit befasst sich mit indoor-Algorithmen, welche Laserscanner zum Abtasten des Umfeldes verwenden. Ausgehend von einem polygonbasierten Ansatz wird ein konkreter Algorithmus entwickelt, welcher durch Einsatz diverser Techniken zur Abstraktion von Messdaten hinsichtlich Performance und Genauigkeit optimiert wird.

Mehmet Bulut

Title of the paper

Analysis and Optimization of Algorithms for Mobile-Systems-Localization using Laser Scanners

Keywords

localization, position estimation, algorithms, optimization, indoor, laser scanner, autonomous, vehicle, robot, mobile, polygon, FAUST, HAW, Hamburg University of Applied Sciences

Abstract

An important subtask in the development of an autonomous robot is to keep track of its current position. Depending on the conditions in the operating site, the available equipment and the mobile platform's mechanical characteristics, different algorithms come into consideration. This paper discusses several techniques for indoor mobile robot position estimation that utilize laser scanners for range measurements. Beginning with a polygon-based concept, a concrete Algorithm will be developed and optimized in terms of performance and accuracy by abstraction of raw measure data.

Danksagung

An dieser Stelle möchte ich mich von Herzen bei allen Professoren und Kommilitonen bedanken, die mich durch mein Studium begleitet und unterstützt haben. Mein besonderer Dank geht an meinen betreuenden Professor Dr. rer. nat. Reinhard Baran, der mir stets gut gelaunt zu jeder Uhrzeit mit hilfreichen Tipps zur Seite stand.

Das Studium an der HAW hat mir, wenn auch stellenweise mit Schweiß verbunden, große Freude bereitet. Ich konnte sowohl auf fachlicher, als auch auf menschlicher Ebene viel dazulernen. Ich habe in dieser Zeit viele interessante Menschen kennengelernt und echte Freunde gefunden. Einem davon, Dennis, danke ich an dieser Stelle für das Aufdecken sprachlicher Fehler und unglücklicher Formulierungen.

Last, but not least danke ich meiner geliebten Familie, die mir während der gesamten Studienzeit Kraft und Motivation gab.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

Mitte des 18. Jahrhunderts begann der Mensch, Maschinen einzusetzen. Sei es zur Steigerung der wirtschaftlichen Effizienz von Produktionsprozessen, für die Verrichtung gefährlicher Arbeiten oder für viele andere Einsatzzwecke. Dieser Trend hat sich bis heute gehalten und mit jedem Technologiesprung eröffneten sich neue Ideen und Wege unsere Gesellschaft weiter zu automatisieren.

Heutige Maschinen brauchen immer weniger menschliche Interaktion, um ihre Arbeit korrekt verrichten zu können. Autonome Systeme sind der nächste große Schritt in Richtung Fusion von Mechanik, Computern, Sensorik und Software für die Entwicklung intelligenter Systeme, welche in der Lage sind nicht nur selbständig in der realen Welt zu agieren, sondern sogar in Regionen vorzustoßen, die dem Menschen teilweise noch verschlossen bleiben.

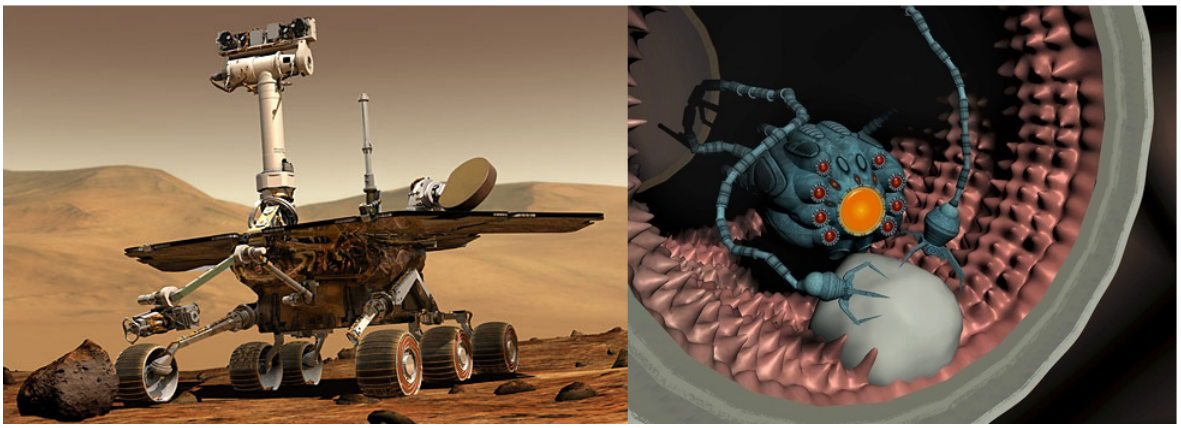


Abbildung 1.1: *links: Der Mars-Rover auf dem Roten Planeten, rechts: Ein Nanoroboter bei Reparaturarbeiten in einer menschlichen Arterie (Tech-Arts)*

1.1 Motivation und Zielsetzung

Eine Aufgabenstellung in der Entwicklung eines sich autonom bewegenden Roboters ist die Bestimmung seiner momentanen Position. Die Algorithmen und Messinstrumente die für

eine präzise und schnelle Orientierung im Raum nötig sind zeigen, mit was für einem hochkomplexen Regelungssystem der menschliche Körper ausgestattet ist.

Was das menschliche Gehirn hier während einer Bewegung in Bruchteilen von Sekunden vollkommen unbewusst mit Hilfe seiner Sinnesorgane vollbringt ist erstaunlich und bewundernswert. Diese Erkenntnis macht das Teilforschungsgebiet der Selbstlokalisierung mobiler Systeme zu einer spannenden und anspruchsvollen Aufgabe.

Unter anderem abhängig von den Gegebenheiten im jeweiligen Einsatzgebiet stehen dabei verschiedene Klassen von Sensoren zur Verfügung. Im Bereich der autonomen Landfahrzeuge haben sich CCD[1]-Kameras, Ultraschall, Infrarot und Laserscanner bewährt. Die einzelnen Sensortypen zeichnen sich durch Unterschiede sowohl in den Möglichkeiten, als auch in den Anforderungen aus, die sie den Computeralgorithmen bieten bzw. an sie stellen.

Die FAUST[2]-Projektgruppe der HAW-Hamburg ist der Oberbegriff für eine Reihe von studentischen Projekten im Bereich der mobilen Robotik. Eines dieser Projekte befasst sich mit der Entwicklung eines autonomen Fahrzeugs basierend auf einem Modell-LKW im Maßstab 1:16 als mobile Plattform.

Das Ziel dieser Abschlussarbeit ist es, verschiedene Algorithmen zur Laserscanner-basierten Positionsbestimmung zu erörtern und Möglichkeiten zur Steigerung von Performance und Auflösung zu erarbeiten. Ausgehend von den Untersuchungsergebnissen soll ein geeigneter Algorithmus als Basis herangezogen und mit Hilfe ausgewählter Optimierungsverfahren für das mobile System “Wedico” einsatzfähig gemacht werden.

1.2 Aufbau dieser Arbeit

Das Kapitel *Grundlagen* vermittelt die für das Verständnis dieser Arbeit nötigen sachlichen Grundlagen. Der Begriff der Ortsbestimmung wird hier klar definiert und von benachbarten Begriffen wie z.B. Ortung abgegrenzt, sowie Merkmale zur Einordnung von Lokalisierungsverfahren geschildert. Ferner werden in der Positionsbestimmung mobiler Systeme verwendete Techniken und Instrumente vorgestellt, sowie deren Schwächen und Problematiken aufgezeigt. Am Ende des Kapitels werden einige geometrische Zusammenhänge aus der Vektoralgebra wiederholt.

Im Kapitel *Analyse* werden die für diese Arbeit geltenden Rahmenbedingungen betrachtet. Dabei wird unter anderem auf die zur Verfügung stehende Hardware eingegangen und

[1] Charge Coupled Device
[2] Fahrerassistenz und autonome Systeme

mögliche Problemstellungen die bei der Integration dieser Komponenten in das Positionsbestimmungssystem auftreten könnten herausgestellt. Daraus, und aus den räumlichen Gegebenheiten werden Anforderungen an die zu entwickelnde Software bestimmt und Methoden zur Umsetzung erarbeitet.

Im darauf folgenden Kapitel *Laserscanner-basierte Lokalisierungsalgorithmen* werden vier Untergruppen vorgestellt, welche auf dem Einsatz von Laserscannern als Sensor basieren. Nach Betrachtung der Gemeinsamkeiten und Unterschiede dieser Algorithmenklassen werden deren Vor- und Nachteile anhand ausgewählter Repräsentanten erörtert. Am Ende des Kapitels wird einer dieser Algorithmen näher analysiert und für den Einsatz im Projekt-LKW in Betracht gezogen.

Das Kapitel *Optimierung* stellt die Probleme des ausgewählten Algorithmus' heraus und zeigt, auf welche Weise das Verfahren für das mobile Systen einsatzfähig gemacht werden kann. Dafür werden verschiedene Optimierungsmaßnahmen vorgestellt und deren Wirkung auf die Laufzeit der Positionsbestimmung dargestellt. Außerdem werden die für die Performance maßgeblichen Parameter erläutert und entsprechende Stellgrößen herausgearbeitet.

In *Design und Realisierung* wird die Architektur des Lokalisierungssystems im Kontext der mobilen Plattform vorgestellt. Danach werden in einer detaillierteren Sicht die Einzelkomponenten und der Prozessablauf beschrieben. Des weiteren werden die Phasen der Positionsbestimmung anhand von Momentaufnahmen aus einem exemplarischen Algorithmenlauf visualisiert und schließlich die bei der Entwicklung eingesetzten Werkzeuge vorgestellt.

Im letzten Kapitel, *Zusammenfassung* werden die Ergebnisse der Arbeit reflektiert und ein Ausblick für zukünftige Arbeiten im Rahmen des "Wedico"-LKW Projektes gegeben.

Hinweise zur Notation

Im Verlauf dieser Arbeit werden die Begriffe *"Ortsbestimmung", "Positionsbestimmung"* und *"Lokalisierung"* bzw. *"Lokalisieren"* synonym verwendet. Mit *"Mobiles System" (MS)* und *"Autonomes System" (AS)* sind autonom agierende Fahrzeuge/Roboter gemeint, wobei letzteres im Kontext der Robotik zu verstehen ist und nicht im Sinne von Rechnernetzen.

2 Grundlagen

Orts- oder Positionsbestimmung, auch Lokalisierung genannt, ist die Ermittlung der eigenen Position relativ zu einem gewissen Bezugspunkt. Der Begriff ist nicht zu verwechseln mit der *Ortung* von (entfernten) Objekten (vgl. Woxikon (2011)).

2.1 Eigenschaften von Lokalisierungssystemen

Systeme zur Positionsbestimmung kommen in vielerlei Szenarien zum Einsatz. Je nach Anwendungsfall werden dabei verschiedene Anforderungen gestellt — so kommen beispielsweise unter Wasser andere Technologien in Frage als an Land. In geschlossenen Räumen eignen sich wieder andere Ansätze als in der Navigation von PKWs. Die vielfältigen Anwendungsmöglichkeiten führten zur Entwicklung einer Vielzahl unterschiedlichster Systeme. Im Folgenden werden einige Merkmale zur Klassifizierung und qualitativen Bewertung von Lokalisierungsverfahren vorgestellt.

2.1.1 Maßstab

Abhängig vom Anwendungsgebiet können Größen wie Reichweite und Auflösung stark voneinander abweichen. Meist stehen diese auch in direktem Zusammenhang zur eigenen Größe.

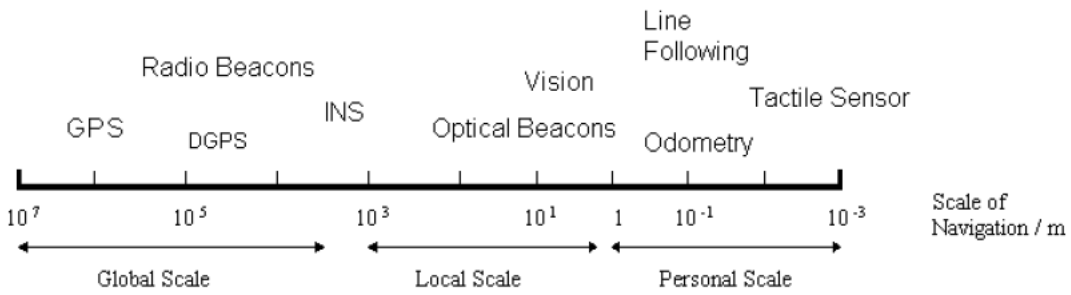


Abbildung 2.1: *Skala zur Einordnung von Positionierungssystemen bezüglich des Maßstabes (Bildquelle: Azenha (2002))*

Nach Azenha (2002) lassen sich Positionierungssysteme bzw. Sensortypen in eine Skala von globalem Maßstab mit $10^4 km$ Reichweite bis in den Millimeterbereich einordnen. Man unterscheidet dabei drei Größenordnungen, *global, local* und *personal*:

Globale Positionierung

Große, weltweit verkehrende Transportmittel wie Schiffe und Flugzeuge navigieren mit globalen Systemen wie dem satellitengestützten GPS[1]. Hierbei ist die Fähigkeit, die aktuelle Position auf der Weltkarte bestimmen zu können ausschlaggebend. Es kommt aber nicht darauf an, die Position auf den Millimeter genau zu bestimmen — eine Genauigkeit von 10 Metern reicht dort meist aus.

Lokale Positionierung

Das Ziel lokaler Positionierungssysteme ist nicht die absolute Bestimmung des Standortes auf der Erde, sondern das Lokalisieren innerhalb eines engeren Kontextes. Statt großer Reichweite geht es hier um eine möglichst hohe Genauigkeit der Positionsdaten. Als Beispiel sei ein autonomes Flurförderfahrzeug in einer 20 mal 20 Meter großen Lagerhalle, welche sich in der Alexanderstr. 52, in 20099 Hamburg befindet. Auf 10 Meter genau sagen zu können, dass sich das Fahrzeug in einer geographischen Lage von 53.5561° Nord und 10.0199° Ost aufhält ist offensichtlich sinnlos. Vielmehr ist hier eine Auflösung der Position im Zentimeter-/Millimeterbereich innerhalb einer auf die Halle beschränkten Weltkarte erforderlich — dass sich das mobile System in Hamburg befindet ist dabei völlig uninteressant.

Personal Positioning

Kommen zu den lokalen Positionsdaten noch Informationen zu Lage (Winkel) und Stellung des autonomen Fahrzeugs hinzu spricht man vom *personal positioning*. Dies ist vor allem dann wichtig, wenn das Fahrzeug aus mehreren beweglichen Teilen zusammengesetzt ist oder in Interaktion mit anderen Objekten steht.

Oft ist es erforderlich verschiedene Positionierungssysteme miteinander zu kombinieren. So reicht bespielsweise im Flugverkehr GPS für die Navigation vom Start- zum Zielort aus, für eine genaue Positionierung innerhalb des Anflugkorridors bei einer Landung jedoch nicht. Hierfür existiert unabhängig vom GPS das Instrument Landing System (ILS) welches eine genauere Lokalisierung bietet (vgl. Wood (1999)).

[1] Global Positioning System

2.1.2 Robustheit

Abhängig vom Einsatzgebiet schwanken die Anforderungen bezüglich der Störanfälligkeit. So muss ein in geschlossenen Räumen arbeitender Reinigungsroboter nicht mit plötzlichem Graupelschauer oder atmosphärischen Störungen rechnen. Dafür wird die Ortsbestimmung im Indoorbereich aber durch andere Dinge erschwert, wie z.B. durch herumstehende Objekte oder stark streuende Wände. Meist sind Lokalisierungssysteme also auf bestimmte Umgebungen spezialisiert und rechnen nur mit den Störgrößen, denen sie auch tatsächlich ausgesetzt sind (vgl. Pinl (2004)).

Ein weiterer Aspekt ist die maximale Anzahl gleichzeitig lokalisierbarer mobiler Systeme. Faktoren wie Abschattung und Überlagerung von Signalwellen durch andere Fahrzeuge können zu fehlerhaften Ergebnissen führen und im schlimmsten Fall das ganze System zusammenbrechen lassen.

2.1.3 Echtzeitfähigkeit

Der Begriff der Echtzeitfähigkeit beschreibt die Fähigkeit eines Systems, eine Aufgabe innerhalb eines definierten Zeitraums durchführen zu können. Man spricht auch von zeitlichem Determinismus. Das Einhalten solcher Zeitschranken ist abhängig vom Anwendungsgebiet von unterschiedlich großer Bedeutung (vgl. Pinl (2004)).

Soft Realtime

Bei weichen Echtzeitanforderungen ist die Einhaltung der Zeitschranken zwar wünschenswert, jedoch für das korrekte Arbeiten des Gesamtsystems nicht ausschlaggebend. Ein Beispiel hierfür wäre das Reaktionsverhalten der graphischen Benutzeroberfläche eines Computerprogramms. Es ist zwar gewünscht dass das Programm auf Benutzereingaben sofort reagiert, jedoch führt es zu keinem Fehlverhalten sollte sich die Reaktion einmal um wenige Millisekunden verzögern.

Hard Realtime

Hartes Echtzeitverhalten ist oftmals in industriellen Anwendungen (Regelungstechnik) gefordert. Nicht-Einhalten der Deadlines macht etwaige Rechenergebnisse unbrauchbar, führt zu kritischem Fehlverhalten des Systems und kann sogar zum Totalausfall führen. Beispiele hierfür sind u.a. Motorsteuerungen, Avioniksysteme oder Reaktorsteuerungen.

2.2 Koppelnavigation

Koppelnavigation (engl. *dead reckoning*) ist die laufende Positionsbestimmung eines MS mit Hilfe intern verfügbarer Daten. Nach der Initialisierung der Startposition wird die Nachfolgeposition durch Integration der internen Daten über die Zeit berechnet.

2.2.1 Odometrie

Als Odometrie (griech. *hodós*, "Weg" und *métron*, "Maß") wird die Ortsbestimmung eines MS anhand der Daten seines Vortriebssystems bezeichnet. Über die Raddrehzahl (bei bekanntem Radius) kann auf die Geschwindigkeit geschlossen, und durch zeitliche Integration die gefahrene Strecke berechnet werden. Falls die Radrotation aus bautechnischen Gründen nicht gemessen werden kann, ist eine indirekte Lösung über die Motordrehzahl und Getriebeübersetzung möglich, wodurch der Fehler allerdings größer wird.

Die Bewegungsrichtung kann durch Überwachen des Lenkwinkels ermittelt werden. Eine andere Möglichkeit hierfür ist die differentielle Odometrie, hier werden die linken Räder unabhängig von den rechten Rädern vermessen. Durch die unterschiedlichen Messwerte kann auf gefahrene Kurven geschlossen werden.

In Borenstein (1998) ist eine Übersicht über systematische und nicht-systematische Fehler, welche beim Einsatz von Odometern auftreten können gegeben:

Systematische Fehler	**Nicht-systematische Fehler**
a) ungleiche Raddurchmesser	a) unebene Fahrbahn
b) mittlerer Raddurchmesser entspricht nicht dem Nennwert	b) unvorhergesehene Objekte auf der Fahrbahn
c) Räder nicht 100% parallel	c) Radschlupf
d) Ungenauigkeit im Radstand	d) äußere Krafteinwirkungen
e) begrenzte Encoder-Auflösung	e) interne Krafteinflüsse
f) begrenzte Encoder-Samplingrate	f) Räder haben keinen dedizierten Kontakt zur Fahrbahn

Tabelle 2.1: *Systematische und nicht-systematische Odometrie-Fehler*

2.2.2 Trägheitsnavigation

Alternativ zur Odometrie können beim *dead reckoning* sogenannte Trägheitsnavigationssysteme (engl. *inertial navigation systems (INS)*) eingesetzt werden. Bei diesen Verfahren wer-

den mittels Gyroskopen kontinuierlich Beschleunigungen in x-, y- und z-Richtung gemessen. Aus diesen Informationen kann auf Geschwindigkeit, Orientierung und dreidimensionale Position geschlossen werden (vgl. Christensen und Fogh (2007)).

Aufgrund der hohen Kosten kommen IN-Systeme heute vorwiegend nur in der Luft- bzw. Raumfahrt oder in Unterseebooten zum Einsatz. Ein Vorteil gegenüber Odometern ist die Unabhängigkeit von der Bodenbeschaffenheit (Radschlupf).

Die Genauigkeit nimmt bei beiden Verfahren mit zunehmender Distanz ab, auch wenn der Fehler bei IN-Systemen vergleichsweise gering ist (Anfang der 1970er Jahre max. 10 nautische Meilen Abweichung in 5 Stunden Flug, vgl. Rells (1978)). Ein weiterer gemeinsamer Schwachpunkt ist, dass die Position immer nur relativ zu einem im Vorfeld bekannten Startpunkt berechnet werden kann. Oftmals sind *dead reckoning*-Systeme Bestandteil integrierter Navigationssysteme, in welchen die errechneten Positionsdaten als Anhaltspunkt für präzisere Algorithmen dienen, oder einen vorübergehenden Ausfall des Primärsystems (z.B. für GPS während einer Tunnelfahrt) kompensieren.

2.3 Weltmodell

Die in Abschnitt 2.2 vorgestellten Verfahren basierten bis auf den gegebenen Startpunkt ausschliesslich auf interne Berechnungen, ohne Möglichkeit auf Überprüfung der auf diese Weise ermittelten Positionsdaten. Stehen jedoch zusätzlich geeignete Sensoren für die Wahrnehmung der Umwelt zur Verfügung, können diese zur Kartographierung und Orientierung genutzt werden. Somit kann ein MS seine Position relativ zu real existierenden Objekten bestimmen, was eine höhere Präzision ermöglicht.

Natürliche Landmarken

Vor der Erfindung der Satellitennavigation nutzten bspw. Seeleute die Sterne, um Messfehler beim *Koppeln* zu korrigieren. Als natürliche Landmarken bezeichnet man gut sichtbare Umgebungsmerkmale, die zur Positionsbestimmung tauglich sind, ohne bewusst für diesen Zweck installiert worden zu sein. Das können im outdoor-Bereich z.B. Bäume, Hügel und Gebäude sein. In geschlossenen Räumen würde man Strukturen wie Wände, Türen oder Gegenstände als natürliche Landmarken bezeichnen (vgl. Kind (2003)).

Künstliche Landmarken

Bei Navigation anhand künstlicher Landmarken werden Objekte speziell zur Positionsbestimmung in der Umwelt installiert.

passive Landmarken sind Objekte die lediglich der Reflektion von Impulsen dienen. Dadurch dass sie eigens für diesen Zweck entwickelt und an günstigen Positionen installiert werden, weisen sie bessere Reflektionseigenschaften auf als natürliche Landmarken (welche auch passiv sind).

aktive Landmarken senden selbst Signale aus oder kommunizieren mit ihrer Umwelt. Eine Variante sind die sogenannten kooperativen elektronischen Transponder, welche dem zu lokalisierenden MS per Funk Rückmeldung darüber geben, wenn sie von dessen Laserstrahl erfasst wurden.

2.4 Vektorrechnung

Im Folgenden werden einige Rechenregeln aus der Vektorrechnung, die bei dem später vorgestellten Lokalisierungsverfahren zum Einsatz kommen, kurz wiederholt.

Ein Vektor im $\mathbb{R}^2$ kann je nach durchzuführender Rechenoperation vorzugsweise in karthesischen Koordinaten (Translation durch einfache Vektoraddition)

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

oder in Polarkoordinaten

$$\vec{v} = (r, \phi)$$

dargestellt werden (Rotation durch Addition von $\Delta\phi$ auf ϕ). Alternativ kann für die Verdrehung eines Vektors in karthesischen Koordinaten auch die folgende Rotationsmatrix angewendet werden:

$$rotate(\vec{v}, \phi) = \begin{pmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{pmatrix} \begin{pmatrix} v_x \\ v_y \end{pmatrix} \tag{2.1}$$

Für die Umrechnung zwischen beiden Formen gilt:

$$r = \sqrt{v_x^2 + v_y^2}$$
$$\phi = arctan\left(\frac{v_y}{v_x}\right)$$
$$v_x = r \cdot cos(\phi)$$
$$v_y = r \cdot sin(\phi)$$

Abstandsberechnung

Für die Berechnung des Abstandes eines Punktes P von einer durch zwei Punkte A und B aufgespannten Geraden, wird zunächst der Vektor $\vec{r} = \vec{B} - \vec{A}$ bestimmt. Danach betrachtet man den Vektor $\vec{d} = \vec{P} - \vec{A}$. Zeigt dessen Projektion $\vec{d'}$ in die selbe Richtung wie $\vec{r}$, d.h. das Skalarprodukt zwischen $\vec{d}$ und $\vec{r}$ ist positiv:

$$\vec{d} \bullet \vec{r} = d_x \cdot r_x + d_y \cdot r_y \geq 0$$

und ist

$$|\vec{d'}| \leq |\vec{r}|$$

so liegt P senkrecht über (oder unter) $\vec{r}$.

Der Abstand a kann dann über das Vektorprodukt berechnet werden:

$$a = \frac{|d_x \cdot r_y - d_y \cdot r_x|}{|\vec{r}|} \tag{2.2}$$

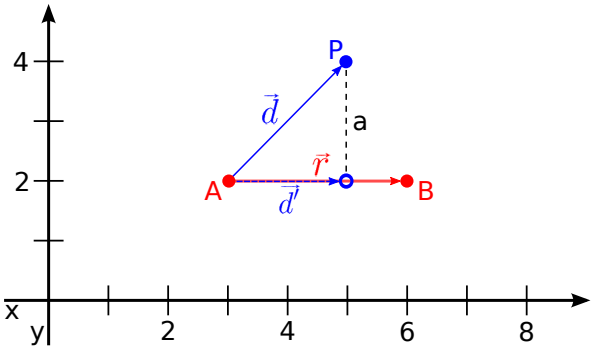


Abbildung 2.2: *Abstandsberechnung zwischen einem Punkt und einer Geraden*

3 Analyse

3.1 Aufgabenstellung

Der Fokus dieser Arbeit liegt auf Indoor-Ortsbestimmungsverfahren, welche Laserscanner zur Vermessung der Umwelt einsetzen. Besonderes Augenmerk wird dabei auf Echtzeiteigenschaften und Performance gelegt.

Die Aktualisierungsrate und Echtzeitfähigkeit eines Positionierungssystems hat direkten Einfluss auf die maximal mögliche Fortbewegungsgeschwindigkeit eines MS. Diese ist wiederum zum einen von der für die Lokalisierungsalgorithmen benötigten Rechenzeit abhängig, zum anderen spielt die von den eingesetzten Sensoren benötigte Messdauer eine wichtige Rolle und beschränkt die maximale Aktualisierungsrate des Systems.

In diesem Kapitel soll die im Rahmen des “Wedico”-Projektes zur Verfügung stehende Hardware und Einsatzumgebung untersucht werden. Die Ergebnisse dieses Kapitels dienen dazu, mögliche Problemstellungen beim Einsatz der Sensorik festzustellen und geeignete Gegenmaßnahmen zu entwickeln.

3.2 Mobile Plattform

Als mobile Plattform kommt ein Modell-LKW des Typs “Wedico *Peterbilt 359*” im Maßstab 1:16 zum Einsatz. Es handelt sich dabei um einen 3-achsigen Sattelschlepper mit 3-Gang-Getriebe und 12V-Elektromotor.

Abbildung 3.1: *Peterbilt 359 (1:16)*

Hinter der Fahrerkabine des LKW wird ein Laserscanner der Firma *Hokuyo Automatic* montiert. Als Zielsystem für die Positionsbestimmung dient ein Lenovo X300 Laptop, welcher auf einem offenen Auflieger im Sattelzug transportiert wird.

3.3 Laserscanner

Der URG-04LX-UG01 von dem japanischen Hersteller Hokuyo ist ein low-cost Indoor-LIDAR[1] und wurde speziell für die Zielgruppe Studenten im Robotikbereich entwickelt.

Abbildung 3.2: *URG-04LX-UG01 LIDAR*

[1] LIght Detection And Ranging

Der Hokuyo LIDAR gehört zur Kategorie der AMCW[2]-Sensoren. Abb. 3.3 zeigt das Arbeitsschema:

1. Der Laser gibt einen Infrarot-Laserstrahl ab, dessen Richtung mittels eines rotierenden Spiegels verändert wird.
2. Wenn der Laserstrahl auf ein Objekt trifft wird er reflektiert. Die reflektierten Strahlen werden im Gerät wieder durch rotierenden Spiegel umgelenkt und von einer Photodiode aufgenommen.
3. Durch Phasenvergleich von ausgesendeter und empfangener Strahlung wird der Abstand vom Laserscanner zum Objekt berechnet.

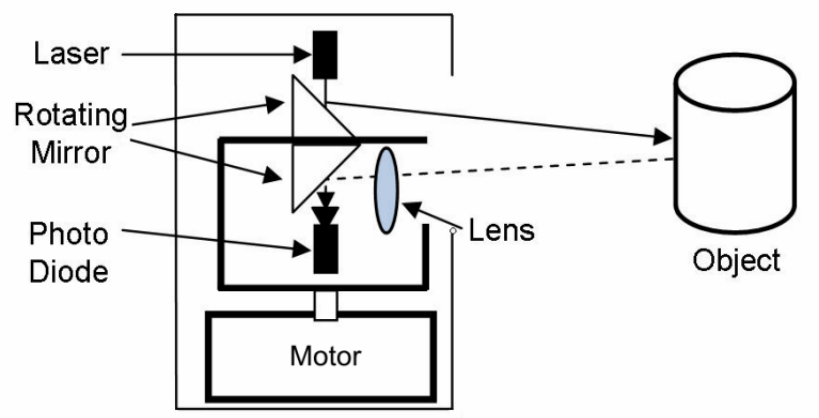


Abbildung 3.3: *Arbeitsweise der URG-04LX Serie (Bildquelle: Okubo u. a. (2009))*

3.3.1 Technische Spezifikation

Die angegebene Scangeschwindigkeit (siehe Tabelle 3.1) ist mechanisch durch die Rotationsgeschwindigkeit des Spiegels (600 U/min) auf 100ms pro Scan begrenzt:

$$\frac{600U}{60s} = 10U/s \Rightarrow 10Hz\ Scanrate \tag{3.1}$$

Die vom Hersteller angegebene Scanrate von 10Hz konnte bei einer Studie dieser Geräteklasse allerdings nicht bestätigt werden. Das Mittel über 1000 Scans erbrachte eine durchschnittliche Messdauer von 109ms (9.17Hz) (vgl. Okubo u. a. (2009)).

[2] Amplitude Modulated Continuous Wave

Im Vergleich zu Laserscannern anderer Hersteller wie der Firma SICK AG sind die Hokuyo LIDARs der UG-04LX-Serie unterlegen was Scangenauigkeit und Reichweite betrifft. Ihr Vorteil liegt jedoch in ihrer kompakten und leichten Bauweise, sowie der geringen Betriebsspannung von 5V. Das bedeutet in der Praxis, dass der Hokuyo keine separate Spannungsquelle benötigt und einfach über die USB-Schnittstelle eines Laptops betrieben werden kann.

Stromversorgung	5V DC (über USB)
Lichtquelle	Halbleiter-Laserdiode (α=785ηm), Laser Klasse 1
Messbereich	20 – 5600mm
Genauigkeit	60 – 1000mm: $\pm$ 30mm 1000 – 4095mm: $\pm$ 3%
Winkel	240°
Winkelauflösung	0.3515625°
Geschwindigkeit	100ms pro Scan
Schnittstelle	USB 2.0 (Mini B)
Gewicht	160g

Tabelle 3.1: *Technische Daten URG-04LX-UG01*

Abb. 3.4 zeigt den Messbereich des Laserscanners. Es werden 769 Messungen in 0.36°-Schritten durchgeführt, wovon die mittleren 682 im gültigen Intervall [-120°,+120°] liegen.

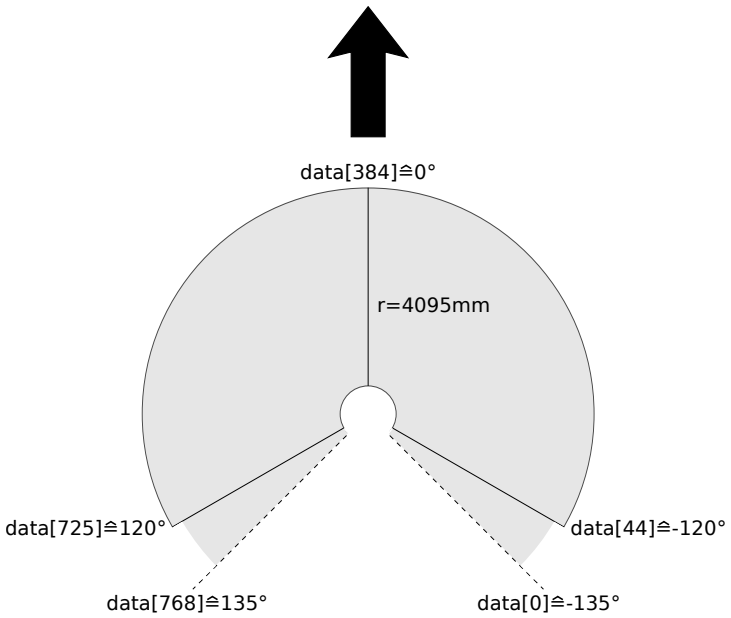


Abbildung 3.4: *Messbereich und Messdaten-Indizierung*

3.3.2 Messungen bei Fahrt

Bei einer Messung, die während einer Fortbewegung der mobilen Plattform stattfindet, kann es zu Abweichungen innerhalb einer Messreihe kommen. Die zeitliche Verzögerung zwischen dem ersten Messpunkt bei -120° und dem letzten Messpunkt bei +120° (siehe Abb. 3.4) beträgt

$$\frac{100ms}{360°} \cdot 240° = 66.\overline{6}ms$$

Unter der Annahme, dass sich das Fahrzeug mit einer konstanten Geschwindigkeit von 1ms^{-1} fortbewegt, verschiebt sich der Ursprung der Messungen vom ersten Messpunkt zum letzten Messpunkt um

$$\frac{1m}{1000ms} \cdot 66.\overline{6}ms \approx 67mm$$

Allgemein gilt für den Versatz $|\vec{s}|$ zwischen zwei Messpunkten ϕ_1 und ϕ_2 bei einer Geschwindigkeit von x ms^{-1}

$$|\vec{s}| = \frac{0.1x \cdot |\phi_2 - \phi_1|}{2\pi} m \tag{3.2}$$

Bei der Korrektur des Versatzes wird davon ausgegangen, dass sich das Fahrzeug zwischen den beiden Messpunkten ϕ_1 und ϕ_2 auf einer Geraden entlangbewegt. Auf eine aufwändige Berechnung der zurückgelegten Strecke durch Odometrie wurde nach einer Fehlerabschätzung für das verwendete Verfahren bewusst verzichtet. Abb. 3.5 zeigt schematisch den maximalen Fehler, der durch Gleichung 3.2 unter maximalem Lenkwinkel und maximalem Abstand zwischen ϕ_1 und ϕ_2 verursacht wird.

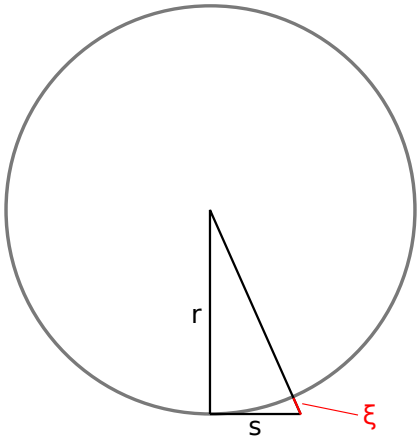


Abbildung 3.5: *Rechenfehler bei maximalem Lenkwinkel*

Der minimale Wendekreis der mobilen Plattform beträgt 1800mm, bei einer Geschwindigkeit

von $1 ms^{-1}$ folgt hieraus für den maximalen Fehler also:

$$\begin{aligned} \xi_{max} &= \sqrt{r_{min}^2 + |s|_{max}^2} - r_{min} \\ &= \left[\sqrt{900^2 + 66.7^2} - 900\right] mm \\ &= 2.47 mm \end{aligned} \tag{3.3}$$

Dieser *worst-case*-Fehler ist vernachlässigbar klein, Gleichung 3.2 liefert für die Praxis ausreichend gute Ergebnisse. Mit steigender Geschwindigkeit nimmt der maximale Fehler zwar zu, jedoch sind Hochgeschwindigkeits-Kurvenfahrten bei der verwendeten mobilen Plattform nicht vorgesehen. Der Einsatz eines odometrischen Verfahrens zur Korrektur des Messversatzes für jeden der 682 Messwinkel würde mehr Rechenzeit kosten als praktischen Nutzen bringen und wird daher nicht weiterverfolgt.

Beispiel

Abb. 3.6 stellt den Sachverhalt anhand einer groben (nur vier Messpunkte) Rundum-Vermessung eines quadratischen Raumes noch einmal in anschaulicher Form dar.

Auf der linken Seite sind die einzelnen, an verschiedenen Orten aufgenommenen Messwerte zu sehen, welche während einer Bewegung ($\vec{s}$) gesammelt wurden. Zum Zeitpunkt der letzten Messung $\vec{m_4}$ hat das Fahrzeug bereits eine Strecke von $3 \cdot |\vec{s}|$ zurückgelegt, was zu einem falschen Abbild der Umgebung führt.

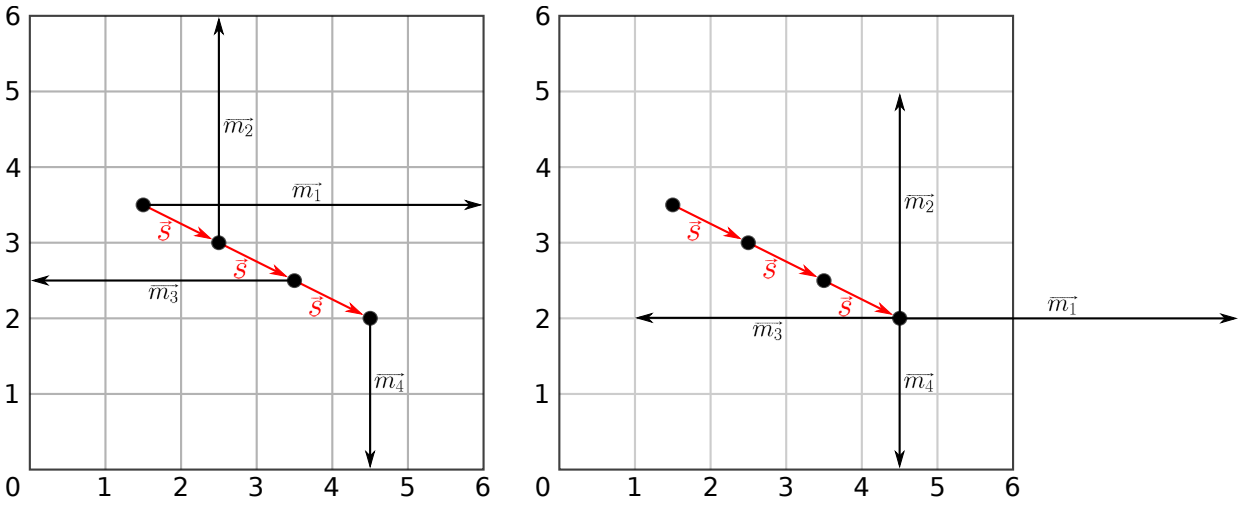


Abbildung 3.6: *links: 360°-Scan bei Fahrt (90° Winkelschrittweite) und fehlerhafte Interpretation der Messung (rechts)*

Mit 3.2 lässt sich der Versatz zwischen zwei Messwerten $\overrightarrow{m_1}$ und $\overrightarrow{m_2}$ beheben. Angenommen der Ursprung von $\overrightarrow{m_2}$ soll als Referenzpunkt für beide Vektoren gelten, so wird $\overrightarrow{m_1}$ transformiert nach:

$$\overrightarrow{m_1'} = \overrightarrow{m_1} - \begin{pmatrix} |\vec{s}| \cdot cos\alpha \\ |\vec{s}| \cdot sin\alpha \end{pmatrix} \tag{3.4}$$

wobei α je nach Wahl des Referenzpunktes die (entgegengesetzte) Fahrtrichtung ist.
Für das Beispiel aus Abb. 3.6 werden also die ersten drei Messwerte mit $\overrightarrow{m_4}$ als Referenz folgendermaßen korrigiert:

$$\overrightarrow{m_1'} = \overrightarrow{m_1} - 3 \cdot \vec{s} = \begin{pmatrix} 4.5 \\ 0 \end{pmatrix} - \begin{pmatrix} 3 \\ -1.5 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$$
$$\overrightarrow{m_2'} = \overrightarrow{m_2} - 2 \cdot \vec{s} = \begin{pmatrix} 0 \\ 3 \end{pmatrix} - \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} -2 \\ 4 \end{pmatrix}$$
$$\overrightarrow{m_3'} = \overrightarrow{m_3} - 1 \cdot \vec{s} = \begin{pmatrix} -3.5 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} -4.5 \\ 0.5 \end{pmatrix}$$

Als Resultat der Transformation bleibt die korrekte Ortsinformation über die erfassten Wände des Raumes erhalten. Zu beachten ist aber, dass die ursprüngliche Winkelschrittweite zwischen den Messwerten verloren geht.

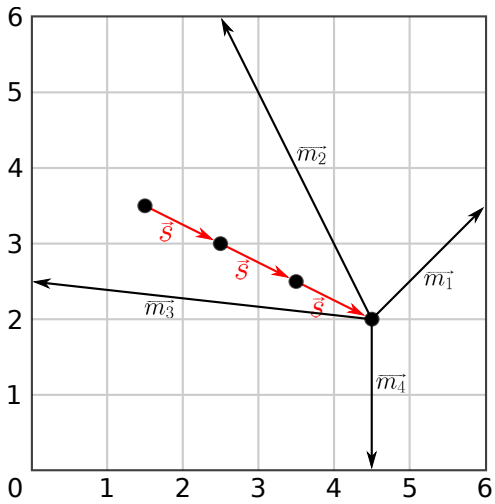


Abbildung 3.7: *Die korrigierten Messwerte zeigen wieder auf die individuellen Orte, welche von den einzelnen Messstationen aus erfasst wurden und geben den Raum somit korrekt wieder.*

3.3.3 Kommunikation

Als Protokoll für die Kommunikation mit einem PC wird SCIP Version 2.0 verwendet (vgl. SCIP2.0). Im Lieferumfang des Gerätes sind neben Gerätetreiber zwei Implementierungen des Protokolls als C und C++ Programmbibliotheken enthalten. Die Übertragung der Messdaten erfolgt stets in ganzen Datenpaketen, d.h. ein Paket enthält immer die Entfernungsdaten einer vollen Messung.

Die Entfernungswerte können wahlweise in 12 bit (max. 4095 mm) oder in 18 bit kodiert werden. Auf diese Weise ist es möglich, die Übertragungsgeschwindigkeit zu erhöhen, wenn nicht mit Entfernungen größer als 4095 mm gerechnet wird.

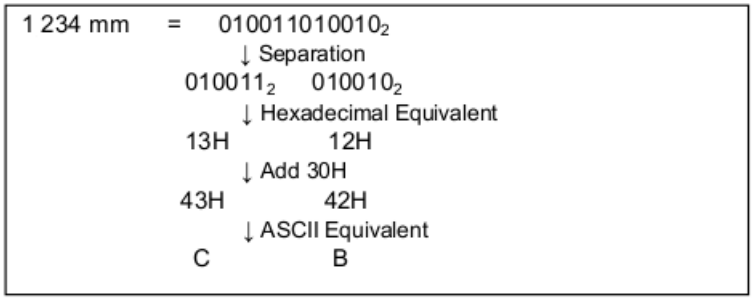


Abbildung 3.8: *Besipiel Two-Character Encoding (Bildquelle: SCIP2.0)*

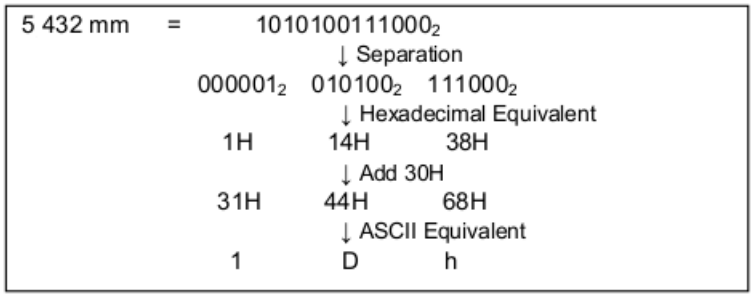


Abbildung 3.9: *Beispiel Three-Character Encoding (Bildquelle: SCIP2.0)*

Abb. 3.8 und 3.9 zeigen das Datenformat für die *Two-* bzw. *Three-Character* Kodierung. In beiden Fällen wird der binäre Entfernungswert (in mm) zunächst in 6 bit lange Segmente aufgeteilt. In einem zweiten Schritt wird auf jedes dieser Segmente der Hex-Wert `0x30` hinzuaddiert.

Die Art der Kodierung lässt sich laut Spezifikation durch verschiedene Arten von Anfragen steuern. Die URG Library des Herstellers hat diese Anfragen-Typen in einem Enumeration-Typ gekapselt. Listing 3.1 zeigt die verschiedenen Request-Arten.

Two-Character Encoding wird durch `URG_GS`, `URG_GS_INTENSITY`, `URG_MS` und `URG_MS_INTENSITY` angefordert und

Three-Character Encoding durch `URG_GD`, `URG_GD_INTENSITY`, `URG_MD` und `URG_MD_INTENSITY`.

Listing 3.1: URG Command Types

```
/**
 * Command type of URG
 */
typedef enum {
    URG_GD,             /* GD command */
    URG_GD_INTENSITY,   /* GD command (including intensity data) */
    URG_GS,             /* GS command */
    URG_MD,             /* MD command */
    URG_MD_INTENSITY,   /* MD command (including intensity data) */
    URG_MS,             /* MS command */
} urg_request_type;
```

Für den Einsatz des mobilen Roboters “Wedico” in seinem relativ großen Testparkour ist das Kommando `URG_MD` sinnvoll, da er dem Laserscanner mitteilt laufend Entfernungsdaten zu messen und diese als 3-Byte Werte zu verschicken.

3.4 Teststrecke

Der Parcours ist in einem der Testräume der FAUST-Projektgruppe in der HAW Hamburg aufgestellt. Es handelt sich dabei um einen quadratischen Rundkurs mit 6 Meter Kantenlänge. Aufgrund der begrenzten Reichweite des Laserscanners (vgl. Tabelle 3.1) wird die Teststrecke zum Raum hin mit Stellwänden abgeschlossen.

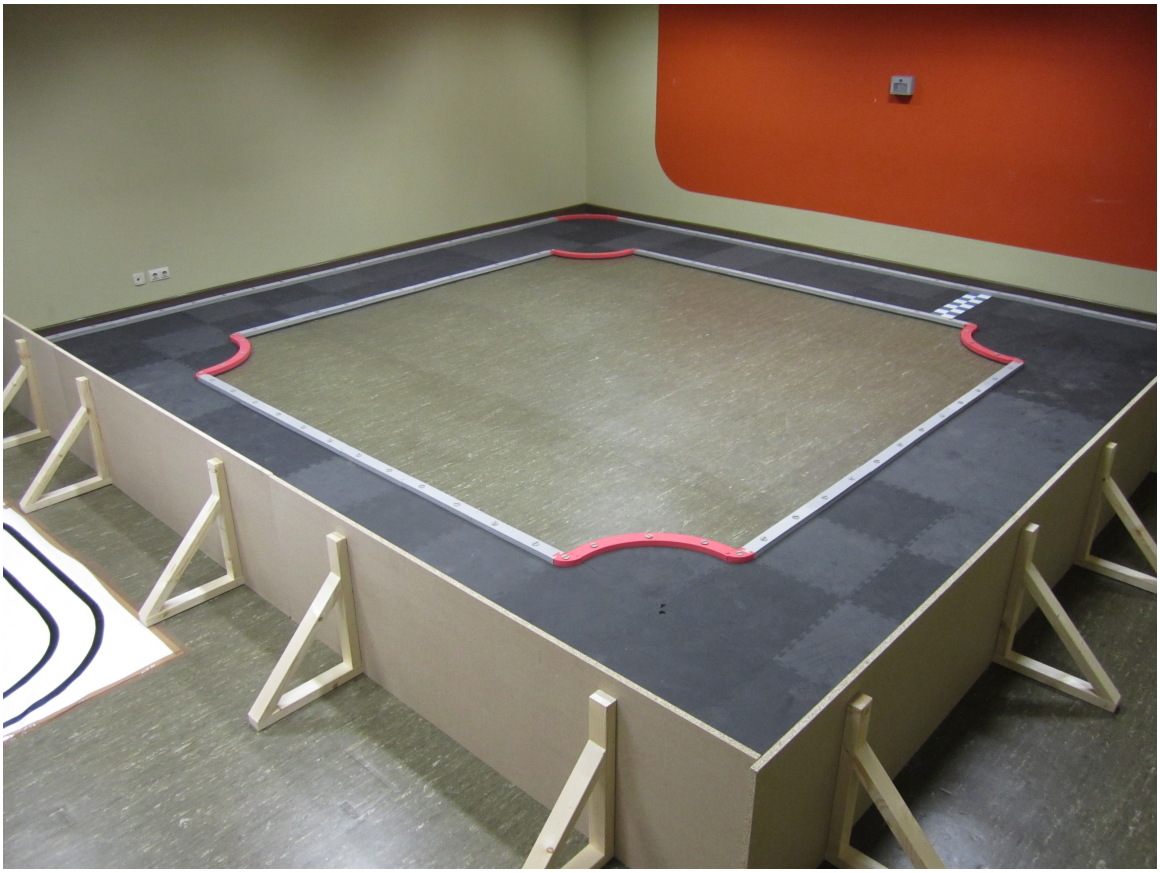


Abbildung 3.10: *Teststrecke*

4 Laserscanner-basierte Lokalisierungsalgorithmen

Algorithmen, die Laserscans zur Bestimmung der Position verwenden, arbeiten auf zwei Mengen von Messwerten aus unterschiedlichen Scans. Der erste Scan (S_{ref}) wird bei der Initialisierung durchgeführt und dient als Referenz für alle weiteren Scans (S) während der laufenden Positionsbestimmung. Für die Bestimmung der Position wird nach jeder Aktualisierung von S (d.h. es sind neue Messdaten verfügbar) versucht, S_{ref} und S zur maximalen Überdeckung zu bringen. Aus der resultierenden Transformation kann im nächsten Schritt die Position des MS relativ zur Startposition (Referenzscan) bestimmt werden.

Bei manchen Algorithmen kann alternativ zu den Referenzscans die Umwelt im Vorfeld manuell ausgemessen und eine Karte daraus erstellt werden. Diese Karte wird dann bei der Initialisierung der Positionsbestimmungsroutine übergeben.

Die Bestimmung der optimalen Transformation kann auf diversen Wegen erfolgen. In diesem Kapitel soll eine Übersicht über vier gängige Konzepte gegeben werden. Diese Konzepte unterscheiden sich in der Art der Interpretation von Messdaten (vgl. Zweigle (WS 2010/11)):

Messpunkte, d.h. die rohen Messdaten, so wie sie aus dem Laserscanner kommen, in Weltkoordinaten übertragen.

Occupancy Grids, d.h. binäre Rasterdarstellung der Umgebung. Jedes Feld im Raster gibt an, ob sich dort ein Objekt befindet oder nicht. Je höher die Auflösung des Rasters, umso realitätsgetreuer wird der Raum abgebildet. Die Speichereffizienz kann gesteigert werden, indem zusammenhängende freie Felder zu großen quadratischen Feldern zusammengefasst werden (*Quadtree*).

Geometrische Repräsentation der Umwelt durch Polygone, Splines oder Kreisbögen. Bei relativ einfachen Konturen ermöglicht dieser Ansatz eine effektive Speicherausnutzung.

Histogramme aus Messdaten über das Vorkommen von x- und y-Koordinaten, sowie von Winkeln.

4.1 *iterative dual correspondence*

Ein Algorithmus, der die Messwerte direkt zur Positionsbestimmung einsetzt ist der IDC[1]-Algorithmus (vgl. Lu und Milios (1997)), welcher zur Klasse der ICP[2]-Algorithmen gehört. ICP-Verfahren ermitteln die aktuelle Position durch Minimierung des Unterschiedes zwischen zwei Punktewolken S_{ref} und S. Das generelle Schema sieht wie folgt aus:

1. Assoziiere jeden Punkt p_i aus S mit einem Punkt p'_i aus S_{ref} nach dem *closest point*-Kriterium (Paarbildung).
2. Berechne die Parameter einer Transformation, welche die Summe der Abstandsquadrate zwischen den Paaren minimiert.
3. Wende berechnete Transformation auf S an.
4. Wiederhole Schritte 1-3 bis das Verfahren konvergiert.

Die Umsetzung des *closest point*-Kriteriums zur Bestimmung korrespondierender Punkte ist offensichtlich der Vergleich aller Punkte p_i mit jedem Punkt aus S_{ref}. Der Punkt mit dem geringsten Abstand wird ausgewählt. Eine obere Grenze, wann zwei Punkte noch benachbart genannt werden dürfen sollte sinnvoll gewählt werden um unrealistische Assoziationen zu vermeiden.

Der IDC-Algorithmus erweitert den ICP-Algorithmus um ein weiteres Korrespondenz-Kriterium, dem *matching range*-Kriterium. Hier wird jedem Punkt p_i derjenige Punkt aus S_{ref} zugeordnet, welcher den gleichen Abstand zum Ort des Laserscanners (p_L) zum Zeitpunkt des aktuellen Scans hat. Folgende Problemstellungen können hierbei in der Praxis auftreten:

1. Kein Punkt aus S_{ref} hat den gleichen Abstand zu p_L wie p_i. Diesen Fall kann man behandeln, indem man S_{ref} nicht Punkt für Punkt durchsucht, sondern immer Punktepaare (p'_i, p'_{i+1}) betrachtet und an geeigneter Stelle interpoliert. Interpolieren ist dann möglich wenn gilt:
   $\left(|\vec{p_i} - \vec{p_L}| > |\vec{p'_i} - \vec{p_L}|\right) \wedge \left(|\vec{p_i} - \vec{p_L}| < |\overrightarrow{p'_{i+1}} - \vec{p_L}|\right)$ oder
   $\left(|\vec{p_i} - \vec{p_L}| < |\vec{p'_i} - \vec{p_L}|\right) \wedge \left(|\vec{p_i} - \vec{p_L}| > |\overrightarrow{p'_{i+1}} - \vec{p_L}|\right)$
   Es wird dann der Punkt auf der Verbindungslinie zwischen p'_i und p'_{i+1} gewählt, der exakt den selben Abstand zu p_L hat wie p_i. Ist auch Interpolieren nicht möglich, wird der Punkt gewählt dessen Abstand dem von p_i am nächsten kommt.

[1] iterative dual correspondence
[2] iterative closest point

2. Mehrere Punkte aus S_{ref} haben den gleichen Abstand zu p_L wie p_i. In diesem Fall kann je nach Implementierung p_i entweder verworfen, oder mit dem p_i' mit dem geringsten Abstand zu p_i gepaart werden.
3. Falsche Zuordnungen können auftreten wenn unterschiedliche Messwinkel die selben Abstandswerte liefern. Der Suchraum (Messwinkel) sollte daher auf ein Minimum der um p_i liegenden Messwerte reduziert werden.

Die beiden Konditionen eignen sich unterschiedlich gut für die Bestimmung der optimalen Transformation. Das *closest point*-Kriterium eignet sich mehr zur Bestimmung der Translation, während das *matching range*-Kriterium bessere Ergebnisse für die Rotation erzielt.

4.1.1 IDC-Algorithmus

Funktionen zur Transformation:

Translation $T_t(S)$, d.h. $\begin{pmatrix} t_x \\ t_y \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix}$ für alle $\vec{p_i}$ aus S

Rotation $R_\phi(S)$, d.h. $\begin{pmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix}$ für alle $\vec{p_i}$ aus S

Nach der Erweiterung von Lu und Milios ergibt sich folgender Ablauf:

1. Initialisierung

   Setze Translationsvektor $\vec{t} := \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ und Rotationswinkel $\phi := 0$
2. Wende $T_t(S)$ und $R_\phi(S)$ an
3. Paarbildung und Berechnung der Transformationsparameter
   a) Wende *closest point*-Kriterium an und bilde Paare $\{(p_1, p_1') \ldots (p_n, p_n')\}$
   Bestimme $\vec{t_1}$ und ϕ_1 so dass die Summe der Abstandsquadrate minimiert wird:

$$E_{dist}(\vec{t_1}, \phi_1) = \sum_{i=1}^{n} \left| \left(R_\phi(\vec{p_i}) + \vec{t}\right) - \vec{p_i'} \right|^2 \Rightarrow minimal$$

   b) Wende *matching range*-Kriterium an und bilde Paare $\{(p_1, p_1'') \ldots (p_n, p_n'')\}$
   Bestimme $\vec{t_2}$ und ϕ_2 so, dass die Summe der Abstandsquadrate minimiert wird:

$$E_{dist}(\vec{t_2}, \phi_2) = \sum_{i=1}^{n} \left| \left(R_\phi(\vec{p_i}) + \vec{t}\right) - \vec{p_i''} \right|^2 \Rightarrow minimal$$

4. Setze $\vec{t} := \vec{t_1}$ und $\phi := \phi_2$
5. Prüfe auf Konvergenz und wiederhole bei Bedarf ab Schritt 2

Nachdem das Verfahren konvergiert, können die Parameter $\vec{t}$ und ϕ für die Positionskorrektur des mobilen Systems verwendet werden. Die Anzahl der Iterationen hängt dabei von den gewählten Schwellwerten ϵ_t und ϵ_ϕ ab.

Bestimmung der Transformationsparameter

Zur Bestimmung der optimalen Transformation gilt es die Summe der Abstandsquadrate aus Schritt 3a) bzw. 3b) des Algorithmus' zu minimieren:

$$E_{dist}(\vec{t}, \phi) = \sum_{i=1}^{n} \left| \left(R_\phi(\vec{p_i}) + \vec{t}\right) - \vec{p_i'} \right|^2 \Rightarrow minimal$$

Für t_x, t_y und ϕ ergibt sich dann nach Lu und Milios (1997):

$$\phi = arctan\left(\frac{S_{xy'} - S_{yx'}}{S_{xx'} + S_{yy'}}\right) \tag{4.1}$$

$$t_x = \overline{x}' - (\overline{x}cos\phi - \overline{y}sin\phi) \tag{4.2}$$

$$t_y = \overline{y}' - (\overline{x}sin\phi - \overline{y}cos\phi) \tag{4.3}$$

mit

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$\overline{x}' = \frac{1}{n}\sum_{i=1}^{n} x_i' \qquad \overline{y}' = \frac{1}{n}\sum_{i=1}^{n} y_i'$$

$$S_{xx'} = \sum_{i=1}^{n} (x_i - \overline{x})(x_i' - \overline{x}_i') \qquad S_{yy'} = \sum_{i=1}^{n} (y_i - \overline{y})(y_i' - \overline{y}_i')$$

$$S_{xy'} = \sum_{i=1}^{n} (x_i - \overline{x})(y_i' - \overline{y}_i') \qquad S_{yx'} = \sum_{i=1}^{n} (y_i - \overline{y})(x_i' - \overline{x}_i')$$

4.1.2 Bewertung

Der IDC-Algorithmus arbeitet direkt auf den Messdaten des Laserscanners. Es wird nicht versucht, diese Daten in irgendeiner Form geometrisch zu interpretieren. Dadurch ist der Al-

gorithmus unabhängig von der Raumgeometrie, d.h. er kann im Gegensatz zu vielen anderen Algorithmen auch in Umgebungen eingesetzt werden, die keine ausreichend polygonale Struktur aufweisen.

Die Kehrseite dieses Ansatzes ist jedoch, dass die Anzahl der Vergleiche für das Ermitteln korrespondierender Punkte mitunter sehr hoch werden kann, da sie direkt von der Anzahl der vorliegenden Messpunkte abhängt. Folgender Pseudocode soll verwendet werden, um die Gesamtkomplexität des Verfahrens abzuschätzen:

```
m := NUM_ITERATIONS
n := NUM_POINTS

for i:=1 to m do
    T_t(S)
    R_φ(S)
    for j:=0 to n-1 do
        for k:=0 to n-1 do
            i_cp := distance_to_index(p_j, p'_k)
            i_mr := range_to_index(p_L, p"_k)
        end for
        (p, p')_j := closest_point(p_j, S_ref, i_cp)
        (p, p")_j := matching_range(p_j, S_ref, i_mr)
    end for
    t += min_error(P, P')
    φ += min_error(P, P")
end for
```

Für Translation und Rotation von S gilt jeweils $O(n)$, zusammen gilt also an der Stelle $O(2n)$. Der teure Abschnitt des Algorithmus', die Stelle wo die korrespondierenden Punkte gesucht werden, hat einen Aufwand von $O(n^2)$ für n Punkte. Zieht man die Anzahl der Iterationen noch hinzu ergibt sich somit ein Gesamtaufwand von $O(m(2n + n^2))$.

4.2 Gitterbasierte Algorithmen

Eine weitere Klasse von Positionsbestimmungsalgorithmen durch paarweise Überdeckung (*feature matching*) von Laserscans stellen jene dar, die die Messwerte eines Scans zunächst als ein binäres Gitter aus belegten und freien Zellen abstrahieren. Dabei wird in jede Zelle an der ein Hindernis detektiert wurde eine 1 und in alle anderen Zellen eine 0 eingetragen. Alle weiteren Operationen erfolgen dann auf diesen Gittern (vgl. Kuipers und Byun (1991)).

Im Folgenden soll anhand einer sehr einfachen Variante das generelle Konzept vorgestellt, sowie dessen Vor- und Nachteile diskutiert werden.

4.2.1 Einfacher *occupancy grid-Algorithmus*

Im Vorfeld werden zunächst Gittergröße und abzusuchender Transformationsraum festgelegt. Der Suchraum, d.h. der Bereich in dem G sukzessive verschoben und verdreht wird sei wie folgt definiert:

$$t_x \in \{x_{min} \dots x_{max}\} \subset \mathbb{N}$$
$$t_y \in \{y_{min} \dots y_{max}\} \subset \mathbb{N}$$
$$\phi \in \{\phi_{min} \dots \phi_{max}\} \subset \mathbb{N}$$

Folgender Pseudocode beschreibt einen einfachen *match*-Algorithmus mit *occupancy grids*:

```
G_ref := create_grid(S_ref, n)
G := create_grid(S, n)
match := G_ref AND G

for φ := φ_min to φ_max do
    G_φ := R_φ(G)
    for t_x:=x_min to x_max do
        G_φx := T_x(G_φ)
        for t_y:=y_min to y_max do
            G_φxy:= T_y(G_φx)
            match' := G_ref AND G_φxy
            if(match' > match) then
                match := match'
                t'_x := t_x
                t'_y := t_y
                φ' := φ
            end if
        end for
    end for
end for
```

Nach jeder neuen Transformation $G_{\phi xy}$ wird durch zellenweises UND der Grad der Übereinstimmung ermittelt. Wurde ein besseres Resultat erzielt als zuvor, werden die aktuellen

Parameter gespeichert. Nach vollständigem Durchlauf des Suchraumes enthalten t'_x, t'_y und ϕ' die gesuchte Transformation.

Abb. 4.1 zeigt Momentaufnahmen eines Algorithmenlaufs für ein Gitter mit n=12, aus Gründen der Anschaulichkeit stark idealisiert dargestellt und ohne Rotation:

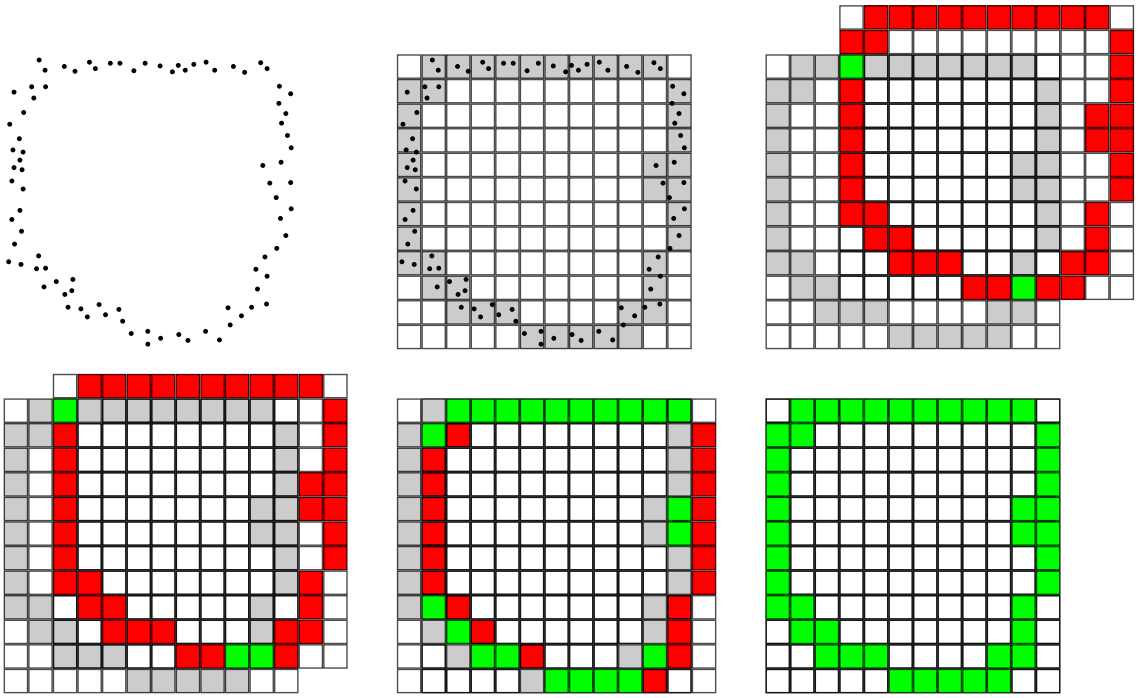


Abbildung 4.1: *Prinzipieller Ablauf eines Occupancy-Grid-Verfahrens*

1. Punktewolke aus Referenzscan.
2. Gitterbildung G_{ref} mit 38 belegten Zellen.
3. Neuer Scan und Gitterbildung G mit anfänglich zwei übereinstimmenden Zellen.
4. G nach Translation um $\begin{pmatrix}-1\\-1\end{pmatrix}$, drei Zellen Überdeckung.
5. G nach weiterer Translation um $\begin{pmatrix}-1\\-1\end{pmatrix}$, 22 Zellen Überdeckung.
6. G nach weiterer Translation um $\begin{pmatrix}-1\\0\end{pmatrix}$, 100% Überdeckung.

Die gesuchte Transformation lautet für diesen idealisiert dargestellten Fall $\vec{t} = \begin{pmatrix} -3 \\ -2 \end{pmatrix}$.

Unter realen Bedingungen nimmt die Wahrscheinlichkeit 100% Übereinstimmung zu erreichen mit fallender Zellgröße ab.

4.2.2 Bewertung

Die Idee, Messwerte auf Raster abzubilden eignet sich vor allem für stark rauschende Umgebungen. Man kann die Zellgröße und Anzahl der Messpunkte, ab wann eine Zelle als belegt zu betrachten ist, dazu nutzen um viele kleine Ausreisser aus der Punktewolke zu eliminieren.

Ein großes Problem dieses Ansatzes ist jedoch die richtige Wahl der Zellgröße: macht man das Gitter zu grob wird die Bestimmung der Pose sehr ungenau, ist das Gitter wiederum zu fein eingestellt nimmt der Rechenaufwand enorm zu. Im Extremfall kann es dazu kommen, dass die Anzahl der Gitterzellen die Anzahl der eigentliche Messwerte übersteigt.

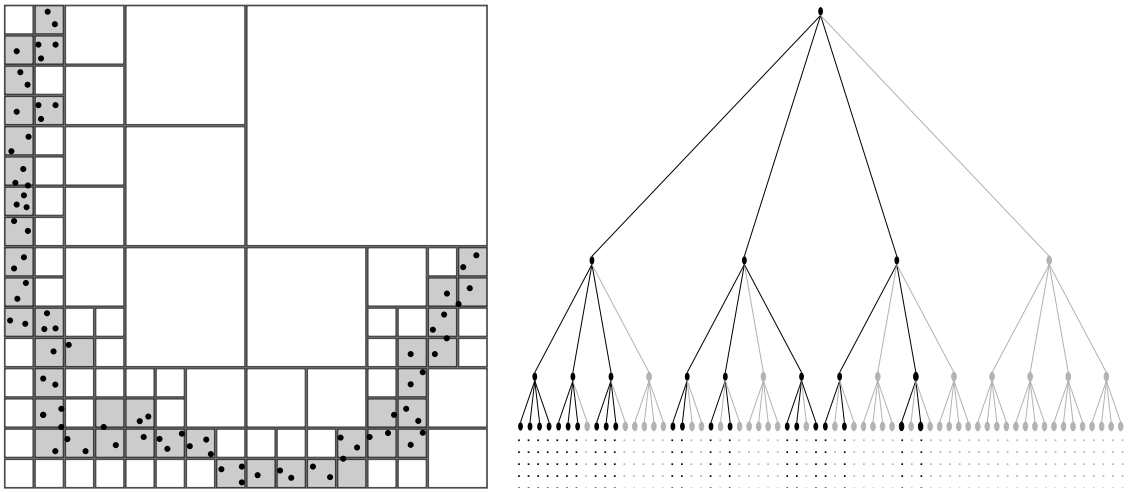


Abbildung 4.2: *Quadtree*

Eine Verbesserung bezüglich Rechenaufwand und Speicherbedarf stellt der Einsatz von Bäumen (Binär-, Vierfach-, Oktalbäume, usw.) als Datenstruktur für das Gitter dar (Abb. 4.2). Durch die Baumstruktur werden zusammenhängende unbelegte Gebiete in einzelne große Zellen zusammengefasst (vgl. Kraetzschmar u. a. (2004)).

Neben der Gittergröße spielen auch die Such-Intervalle für t_x, t_y und ϕ eine Rolle für die Performance des Verfahrens. Seien l_x, l_y und l_ϕ die entsprechenden Intervalllängen und n die Anzahl der Spalten im quadratischen Gitter, so beläuft sich die Gesamtkomplexität

des Algorithmus' aus 4.2.1 auf $O(l_x l_y l_\phi n^2)$. Ein Baum ermöglicht prinzipiell logarithmischen Aufwand — vorausgesetzt er ist nicht voll besetzt, da sonst die Zahl der zu bearbeitenden Zellen der eines einfachen Gitters entspräche. Es hängt also auch davon ab was für Messdaten vorliegen und ob sie beim Erstellen der Bäume gefiltert werden, z.B. durch Festlegen eines Schwellwertes (siehe oben).

4.3 Kreuzkorrelation von Histogrammen

Eine ganz andere Darstellung des Raumes verwendet der Algorithmus von Weiss und v.Puttkammer (1995). Statt die Umwelt nach menschlichem Vorbild geometrisch zu betrachten, werden bei diesem Ansatz die Messdaten des Laserscanners statistisch ausgewertet und versucht durch Korrelation zweier zeitlich aufeinanderfolgender Histogramme Translation und Rotation abzuschätzen.

Annahmen

Der Algorithmus, welcher für den Einsatz in geschlossenen Räumen ausgelegt ist, macht sich die stark polygonale Umgebung dort zunutze. In Gebäuden kann man davon ausgehen, dass bestimmte Regelmäßigkeiten in der Häufung von z.B. Winkeln auftreten.

Eine weitere Annahme, und gleichzeitig auch Schwäche des Algorithmus', ist die Voraussetzung, dass aufeinanderfolgende Messwerte eine mehr oder weniger gerade Linie bilden. Dies ist besonders bei Laserscannern mit einer hohen Winkelauflösung (gekoppelt mit dessen Messungenauigkeit) oft nicht der Fall, da sich die Messwerte abhängig von der Entfernung zu Wänden, etc. mehr oder weniger stark häufen können. An diesen Häufungspunkten herrscht ein unregelmäßiges Hin-und-Her sich teilweise überlagernder Messpunkte, sodass der Zusammenhang Messwinkel $\leftrightarrow$ geometrische Reihenfolge nicht mehr gegeben ist. Es bedarf also unter Umständen einer Vorverarbeitung der Messdaten um sie für den Algorithmus nutzbar zumachen.

Invarianten

Zur Berechnung der Translation in x- und y-Richtung, sowie der Rotation zweier Scans wurden Darstellungen der Messdaten gesucht, die jeweils invariant bezüglich der anderen Operationen sind. Ein Winkelhistogramm, d.h. eine Statistik über auftretende Winkel in der Messung, ist bspw. invariant bezüglich der Translation. Erstellt wird es nach folgendem Schema:

1. Bilde Histogramm für das Winkel-Intervall $[-90°, +90°]$
2. Bilde Vektor $\overrightarrow{P_i P_{i+1}}$ zweier aufeinanderfolgender Messpunkte.
3. Bestimme (spitzen) Winkel des Vektors zur x-Achse ohne dessen Richtung zu beachten und runde ihn zu einem diskreten Wert.
4. Inkrementiere das Histogramm an der Stelle des berechneten Winkels.
5. Wiederhole Schritte 2-4 für alle Messpunkte P_i

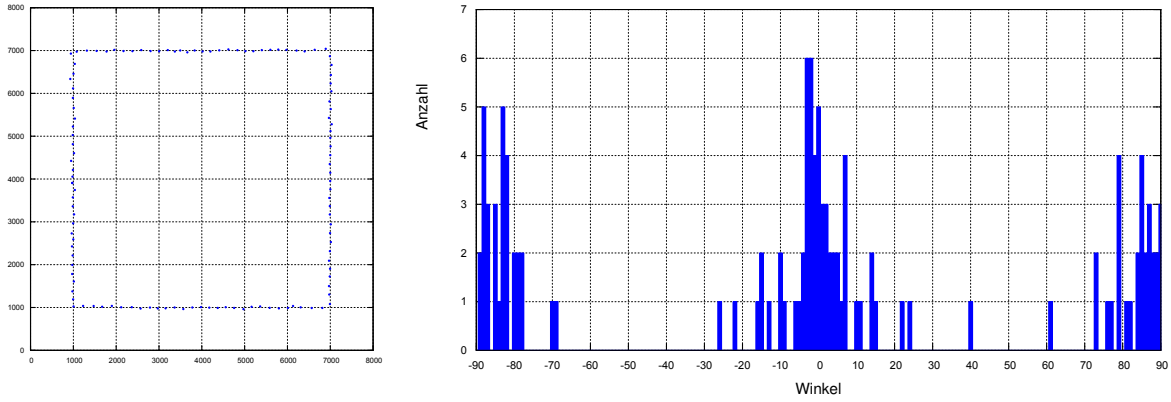


Abbildung 4.3: *Winkelhistogramm (re.) eines Laserscans in einem quadratischen Raum*

Abb. 4.3 zeigt das Winkelhistogramm eines simulierten Scans in einem geschlossenen Raum. Die rechtwinklige Charakteristik des Scanbildes und die senkrechte Orientierung zur x-Achse bleibt erhalten und findet sich in der Häufung von Winkeln um $0°$ und $\pm 90°$ wieder. Man kann sich leicht vorstellen, dass das Histogramm bei einer Verschiebung der Roboterposition erhalten bliebe. Es würde lediglich zu Schwankungen in der Amplitude kommen, bedingt durch die Unterschiedliche Punktedichte durch nähere, oder weiter entfernte Wände. Zu Phasenänderungen kommt es im Histogramm jedoch nur durch Rotation (Abb. 4.4).

Analog werden für die Bestimmung der Translation x- und y-Histogramme gebildet. Ein x-Histogramm ist invariant bezüglich Verschiebungen entlang der y-Achse und umgekehrt.

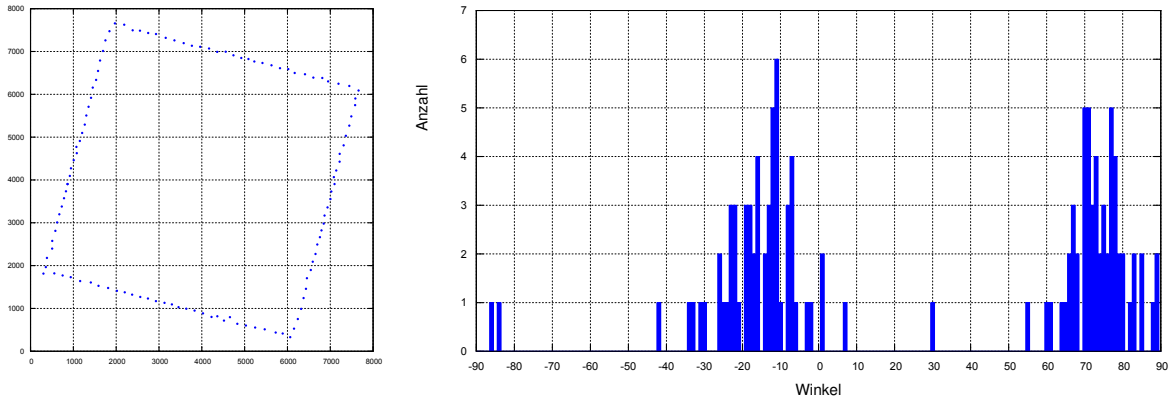


Abbildung 4.4: *Phasenverschiebung des Histogramms durch Rotation um 15°*

4.3.1 Kreuzkorrelationsfunktion

Die Rotation zweier aufeinanderfolgender Scans S_1 und S_2 kann durch Anwenden der Kreuzkorrelationsfunktion auf deren Histogramme h_1 und h_2 berechnet werden:

$$c(j) = \sum_{i=1}^{n} h_1(i) h_2(i+j) \tag{4.4}$$

Die Stelle, an der $c(j)$ sein lokales Maximum annimmt entspricht mit großer Wahrscheinlichkeit der tatsächlichen Verdrehung. Der Definitionsbereich sollte jedoch möglichst schmal (wenige Grad) gewählt werden, da die Funktion sonst mehrere Maxima annehmen kann und eine eindeutige Aussage über die Rotation nicht mehr möglich ist. Da während der Histogrammbildung die Winkel auf diskrete Werte gerundet wurden, sollte jetzt, statt das exakte Maximum zu betrachten, ein Mittelwert über einen engen Bereich um das Maximum herum gebildet werden um die genaue Verdrehung zu ermitteln.

So wie eine Rotation zur Phasenverschiebung im Winkelhistogramm führt, verursacht eine Translation eine Phasenverschiebung im x- bzw. y-Histogramm. Die Translation wird daher analog durch Anwenden der Kreuzkorrelationsfunktion ermittelt.

Gleichung 4.4 ermöglicht in der Form noch keine Aussage über die Güte der Korrelation. Aus diesem Grund wurde sie erweitert zur *normalisierten Kreuzkorrelationsfunktion* mit dem Wertebereich $[-1, 1]$.

$$c_n(j) = \frac{\sum_{i=1}^{n} \left[\left(s_1(i) - \overline{s}_1 \right) \left(s_2(i+j) - \overline{s}_2 \right) \right]}{\sqrt{\sum_{i=1}^{n} \left(s_1(i) - \overline{s}_1 \right)^2 \cdot \sum_{i=1}^{n} \left(s_2(i+j) - \overline{s}_2 \right)^2}} \tag{4.5}$$

mit den Mittelwerten aus n Histogrammzellen:

$$\overline{s}_1 = \frac{1}{n}\sum_{i=1}^{n} h_1(i) \qquad \text{und} \qquad \overline{s}_2 = \frac{1}{n}\sum_{i=1}^{n} h_2(i)$$

Für unkorrelierte Histogramme liefert die Funktion den Wert 0, für gut korrelierte nähert sie sich dem Wert 1.

4.3.2 Bewertung

Der Kreuzkorrelationsalgorithmus ist ein kompakter Indoor-Algorithmus, der ohne viele Schleifen und Iterationen auskommt. Die Güte der Positionskorrektur hängt jedoch stark von den Messdaten ab — das Verfahren liefert nur dann eindeutige und zuverlässige Ergebnisse, wenn die eingehenden Histogramme ausreichend deutliche Maxima aufweisen. Ein Extrembeispiel für eine absolut ungeeignete Umgebung für diesen Ansatz wäre ein runder Raum, da die Winkel hier stets gleichverteilt sind und man nie irgendwelche Phasenänderungen im Histogramm erkennen könnte.

Um Aussagen über die Zeitkomplexität einer konkreten Implementierung des vorgestellten Verfahrens machen zu können, werden wieder einige Zeilen Pseudocode betrachtet. Die Histogramme $h_{\phi 1}$, h_{x1} und h_{y1} aus der jeweiligen Vorrunde werden als Referenzhistogramme in die aktuelle Runde übernommen. Wir stellen also die Histogramme $h_{\phi 2}$, h_{x2} und h_{y2} aus der aktuellen Messung S auf:

```
for i:=0 to n-1 do
    angle := get_angle(S[i])
    h_φ2[angle]++
    h_x2[S[i].x]++
    h_y2[S[i].y]++
end for
```

Die Histogrammbildung kostet wie zu sehen ist $O(n)$ für n Messwerte. Zur besseren Überschaubarkeit wird 4.4 zur Aufwandsabschätzung der eigentlichen Berechnung herangezogen, da die Komplexität beider Funktionen (4.4, 4.5) als gleich betrachtet werden kann.

```
for j:=0 to z-1 do
    for i:=0 to z-1 do
        c_φ[j] += h_φ1[i]*h_φ2[(i+j)%z]
        c_x[j] += h_x1[i]*h_x2[(i+j)%z]
        c_y[j] += h_y1[i]*h_y2[(i+j)%z]
    end for
end for
```

Die Berechnung der Maxima hat $O(z)$, wobei z die Anzahl Zellen im Histogramm ist. Auch wenn sich dadurch eine Gesamtkomplexität von $O(nz^2)$ ergibt kann man in der Regel von relativ schmalen (vgl. 4.3.1) Definitionsbereichen der Kreuzkorrelationsfunktion ausgehen, weshalb der Algorithmus durchaus als schnell angesehen werden kann.

4.4 Überdeckung mit Polygonen

Nachdem in den Abschnitten 4.1 bis 4.3 drei Vertreter unterschiedlicher Klassen vorgestellt wurden, soll im Folgenden ein performance-optimierter Algorithmus zur Positionsbestimmung in geschlossenen Räumen entwickelt werden, welcher auf der Idee der paarweisen Überdeckung von Scans aufsetzt (vgl. Tschisow (2008)).

Bei der paarweisen Überdeckung von Scans existieren zwei Scans: ein Referenzscan S_{ref}, welcher zur Initialisierungszeit aufgenommen und für den restlichen Betrieb des MS beibehalten wird, und n Folgescans S_i, von denen jede einzelne für eine Positionsaktualisierung verwendet und danach vom Folgescan S_{i+1} für die nächste Positionsberechnung überschrieben wird. Die gesuchte Transformation zur optimalen Überdeckung wird ermittelt, indem der Folgescan S_i innerhalb definierter Schranken sukzessive verschoben und rotiert wird. Nach jeder neuen Transformation wird der mittlere Abstand der Scanpunkte aus S_i zum Referenzscan S_{ref} berechnet. Die Transformation, bei der der mittlere Abstand minimal ist, entspricht der gesuchten Transformation, aus welcher dann die Roboterpose abgeleitet werden kann.

Ein Vergleich vom IDC-Algorithmus (vgl. 4.1) und dem *occupancy grid*-Algorithmus aus Abschnitt 4.2 zeigt, dass die Anzahl vorliegender Messdaten bzw. Gitterzellen mehr oder weniger stark ins Gewicht fallen, wenn es um die Laufzeit dieser Algorithmen geht. Für Ansätze wie diesen, welcher den kompletten Scan sehr oft verschiebt, rotiert und darauf arithmetische Operationen ausführt spielt die Menge der zu bearbeitenden Daten eine große Rolle, sodass Wege gesucht werden müssen, Informationen bezüglich der Umwelt zu verdichten.

Polygondarstellung

Eine erste Abstraktion von S_{ref}, welche sich besonders für den Einsatz in Gebäuden anbietet, ist die Darstellung der Umwelt als offener oder geschlossener Polygonzug, d.h. die minimale Menge von Koordinatenpunkten, um den Raum ausreichend genau darzustellen. Dieses Polygon wird durch manuelle Ausmessung erstellt und später dem Algorithmus zur Initialisierungszeit übergeben. An dieser Stelle wird davon ausgegangen, dass solch eine Karte bereits existiert. Wir nennen dieses Polygon fortan P und die Polygonpunkte p_i.

4.4.1 Algorithmus

Zur Bestimmung von Translation und Rotation wird ähnlich zum Beispiel in 4.2 ein Parameterraum für $\vec{t}$ und ϕ festgelegt, den es zu durchsuchen gilt:

$$t_x \in \{x_{min} \dots x_{max}\} \subset \mathbb{N}$$
$$t_y \in \{y_{min} \dots y_{max}\} \subset \mathbb{N}$$
$$\phi \in \{\phi_{min} \dots \phi_{max}\} \subset \mathbb{N}$$

Für jede Transformation $T_t(R_\phi(S))$ wird das Mittel der Abstandsquadrate aller Scanpunkte $s_i \in S$ zum Polygon gebildet. Der Abstand eines Punktes zum Polygon wird definiert als der Abstand zu dem Polygonabschnitt, welcher am nächsten zum Punkt liegt (vgl. 2.4). Die Transformation, bei der der mittlere Abstand minimal wird entspricht mit großer Wahrscheinlichkeit der tatsächlichen Verschiebung und Rotation.

Im Pseudocode lässt sich der Algorithmus folgendermaßen formulieren:

```
avg_min := dist_max

for φ := φ_min to φ_max do
    G_φ := R_φ(G)
    for t_x:=x_min to x_max do
        G_φx := T_x(G_φ)
        for t_y:=y_min to y_max do
            G_φxy:= T_y(G_φx)
            avg_dist := calc_avg_distance(S, P)
            if(avg_dist < avg_min) then
                avg_min := avg_dist
                t'_x := t_x
                t'_y := t_y
                φ' := φ
```

```
            end if
        end for
    end for
end for
```

4.4.2 Analyse

Abb. 4.5 veranschaulicht einen Algorithmendurchlauf zur Bestimmung der Transformationsparameter. In dem Beispiel ist die gesuchte Transformation $\begin{pmatrix} 0 \\ 0 \\ 0^\circ \end{pmatrix}$, dargestellt ist der Verlauf des mittleren Abstandes des Scans abhängig von der Translation in x- und y-Richtung, die Rotationskomponente wurde zur besseren Anschaulichkeit konstant auf 0° gehalten. t_x und t_y bewegen sich dabei im Intervall $[-500, 500]mm$ mit einer Schrittweite von $10mm$. Das Minimum der abgebildeten Oberfläche beträgt $3.2mm$ und befindet sich bei $\vec{t} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, was der gesuchten Translation entspricht.

Zeitkomplexität

Wie bereits beim *occupancy grid*-Algorithmus aus 4.2 stellt das Durchsuchen des gesamten Parameterraumes t_x, t_y und ϕ einen großen Teil der zu verrichtenden Arbeit dar. Seien s_ϕ, s_x und s_y die Schrittweiten in den Intervallen l_ϕ, l_x und l_y, bedeutet das für die Zahl der Verschiebungen und Verdrehungen O(ϕxy), mit

$$\phi = \frac{l_\phi}{s_\phi}$$
$$x = \frac{l_x}{s_x}$$
$$y = \frac{l_y}{s_y}$$

Die Gesamtkomplexität des Algorithmus' kann man demnach mit $O(\phi xynm)$ abschätzen, wobei n die Anzahl Scanpunkte und m die Zahl der Polygonabschnitte ist.

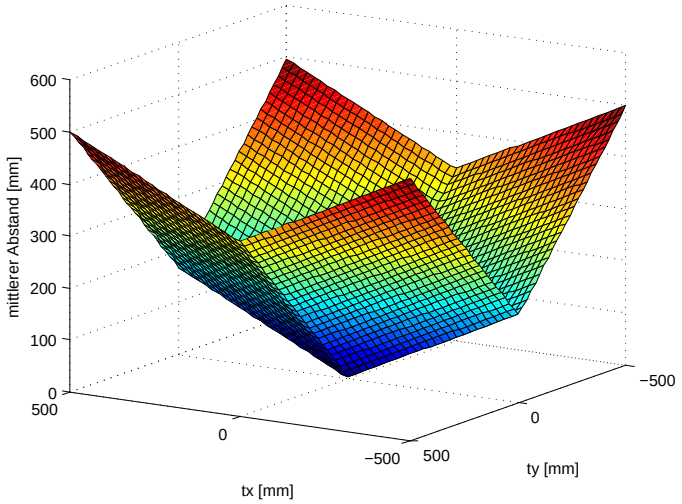


Abbildung 4.5: *Verlauf der mittleren Abstände in Abhängigkeit von Translation*

Bei der Erstellung der Grafik 4.5 wurde die Positionsbestimmung mit einem Polygon ($m = 4$) und einem 240°-Scan ($n = 682$) durchgeführt. Für s_ϕ wurden 11, für s_x und s_y jeweils 101 Schritte festgelegt. Damit gilt für die Zahl der Einzelabstandsberechnungen N_{dist} (d.h. Punkt zu Polygonabschnitt):

$$N_{dist} = 11 \cdot 101^2 \cdot 4 \cdot 682 = 306\,111\,608 \tag{4.6}$$

Zusätzlich zu den Abstandsberechnungen kommt noch der Rechenaufwand hinzu, welcher durch die vielen Verschiebungen und Rotationen hervorgerufen wird.

Das Zahlenbeispiel zeigt, dass die beiden direkt voneinander abhängigen Qualistätsmerkmale Aktualisierungsrate und Auflösung bei diesem Ansatz sehr stark von der tatsächlichen Zahl der nötigen Transformationen und Abstandsberechnungen abhängt. Ohne sinnvolle Maßnahmen, diese Parameter im Vorfeld zu minimieren, ist das Verfahren unbrauchbar langsam und zu ungenau, um tatsächlich in einem mobilen System eingesetzt werden zu können.

Im folgenden Kapitel werden einige algorithmische Optimierungsmaßnahmen zur grundsätzlichen Verringerung der Rechenoperationen vorgestellt.

5 Optimierungsmöglichkeiten

Der in Abschnitt 3.3 vorgestellte Laserscanner liefert Scandaten im 10Hz-Takt. Um die Möglichkeiten des Gerätes voll ausnutzen zu können, muss die von der Lokalisierungs-Software für eine Positionsaktualisierung benötigte Zeit auf gut unter $100ms$ gebracht werden. In diesem Kapitel sollen Möglichkeiten und Wege erörtert werden, wie man den Algorithmus aus 4.4 dahingehend optimieren kann, dass eine höhere Aktualisierungsrate erreicht wird.

Die Rechenzeit ist, wie in Abschnitt 4.4.2 identifiziert, maßgeblich von den folgenden zwei Größen abhängig:

a) die Anzahl der Messdaten

b) die Anzahl der Transformationen

In den folgenden zwei Abschnitten 5.1 und 5.2 werden wir uns mit dem Thema Messdatenreduzierung befassen. Weniger Messdaten bedeutet weniger Abstandsberechnungen und weniger Rechenaufwand bei den wiederholt durchgeführten Transformationen.

Am Ende des Kapitels sollen die Vor- und Nachteile manuell ausgemessenener Weltkarten besprochen und gegebenenfalls alternative Modelle für das Mapping der Umgebung zur Initialisierungszeit entwickelt werden.

Rohe Daten

Der verwendete Laserscanner hat wie bereits erwähnt eine Winkelauflösung von ca. $0.36°$. Über den gesamten Messbereich von $240°$ kommt man dabei auf maximal 682 brauchbare Entfernungswerte mit einem Streufaktor von 3% (Abb. 5.1). In dieser relativ großen Datenmenge gibt es viele Messwerte, die in gewissem Sinne als äquivalent betrachtet werden können. Das gilt insbesondere dann, wenn sich das MS in der Nähe einer Wand oder ähnlichem befindet und sehr viele Messpunkte dicht auf dicht liegen. Sich mehr oder weniger zufällig ein Dutzend Messwerte aus dem Topf herauszunehmen ist sicherlich eine Methode, jedoch wird sie in der Praxis keine brauchbaren Ergebnisse liefern, da man sich dabei auf Werte verlassen würde die mit einen Absolutfehler von bis zu $120mm$ behaftet sein können (vgl. 3.3).

Eine geeignete Vorverarbeitung der Messdaten sollte also die Zahl der Messpunkte verkleinern und dabei die Gesamtaussage der Messung über die Position im Raum bewahren, indem sie z.B. durch Mittelwertbildung die Streuung des Laserscanners ausgleicht.

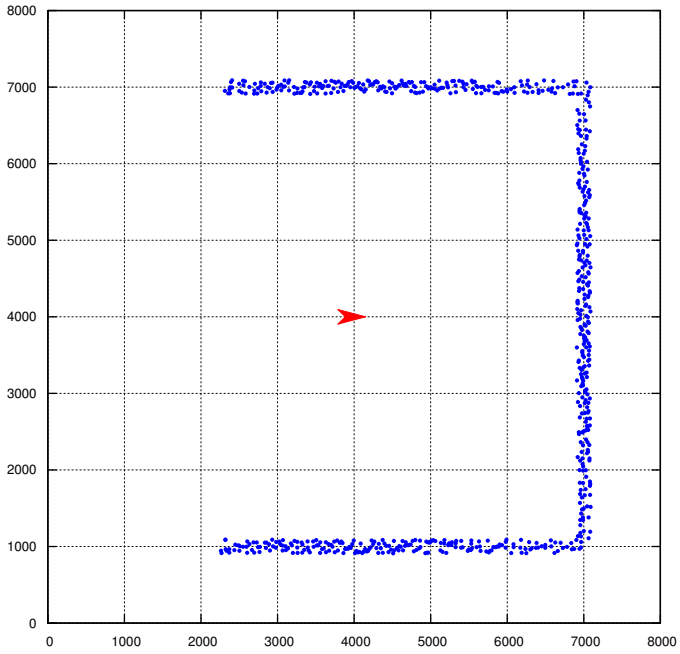


Abbildung 5.1: *240° -Messung in einem simulierten Raum von 6×6m Größe aus Pose* $(4000 \quad 4000 \quad 0°)^T$*. Zur Simulation der Messpunkte wird von der spezifizierten (vgl. Tab. 3.1) Maximalstreuung ausgegangen (3%).*

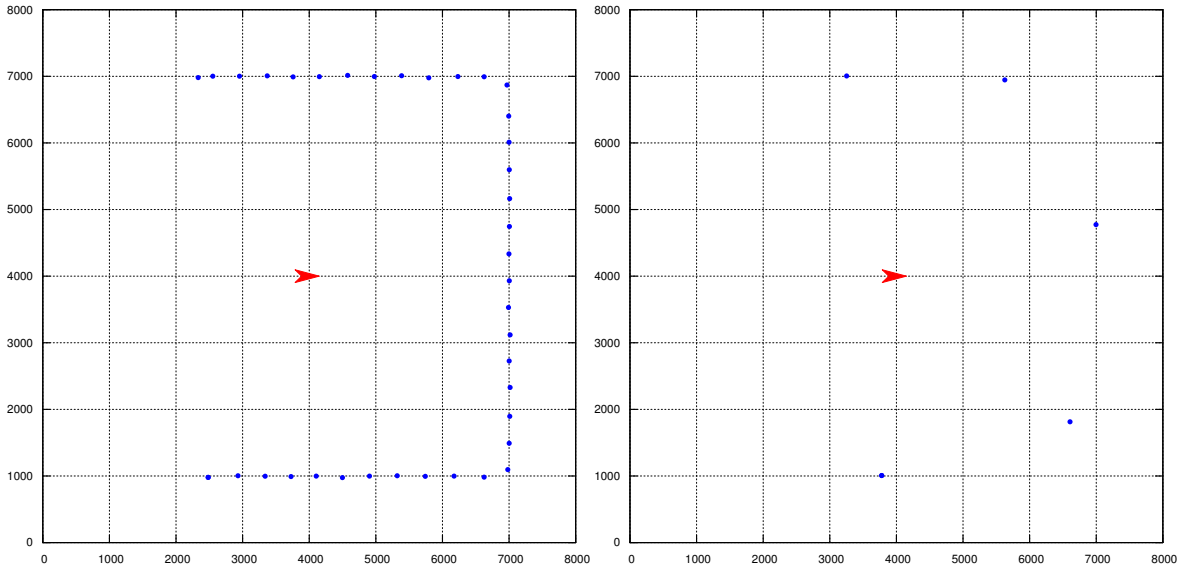
5.1 Reduktionsfilter

Ein Weg, die Datenmenge zu verkleinern ist die Anwendung eines Reduktionsfilters (vgl. Zhang (WS 2010/11)). Die Idee des Filters ist, n benachbarte Messpunkte innerhalb eines definierten Radius' r durch einen neuen Punkt zu ersetzen, welcher dem Mittelwert der ursprünglichen Punkte entspricht. Der gewählte Schwellwert für r bestimmt dabei, wie viele Punkte eliminiert werden. Ein kleiner Wert für r siebt nur wenige Punkte aus der Punktemenge heraus, während ein zu groß gewählter Radius die Charakteristik der Messung so weit verändern kann, dass die erzeugten neuen Punkte die Konturen des vermessenen Raumes nicht mehr korrekt wiedergeben.

Folgender Pseudocode soll beschreiben, wie der Algorithmus eine gegebene Menge an Scandaten S durch Mittelwertbildung reduziert und daraus eine neue Punktemenge S' erzeugt:

```
p0 := S[0]
psum := S[0]
n := 1
for i := 1 to |S|-1 do
    p := S[i]
    if(distance(p0, p) < 2r) then
        psum += p
        n++
    else
        S'[k++] := psum/n
        p0 := p
        psum := p
        n := 1
    end if
end for
S'[k++] := psum/n
```

Der Reduktionsfilter ist eine ebenso einfache wie effektive Methode, die Zahl der Rechenoperationen im eigentlichen Lokalisierungsalgorithmus zu verringern. Der zusätzliche Rechenaufwand durch den Filter beträgt $O(n)$ und fällt im Verhältnis zur eingesparten Rechenzeit nicht auf.



(a) Filterradius r=200 (b) Filterradius r=1500

Abbildung 5.2: *Messdaten nach Anwendung des Reduktionsfilters*

Durch die Akkumulation (mit anschließender Mittlung) aufeinanderfolgender Messpunkte besteht jedoch die Gefahr, dass markante Stellen wie z.B. Ecken “weggeglättet” werden. Abb. 5.2 zeigt die Wirkung des Reduktionsfilters: die Anzahl der Messwerte hat sich von ursprünglich 682 (vgl. Abb 4.5) auf 38 reduziert (a). Deutlich zu sehen auf dem Bild daneben (b) ist die Verfälschung der Messdaten durch zu groß eingestellten Filterradius.

5.2 Linienextraktion

In stark polygonalen Umgebungen macht es Sinn, nach markanten Merkmalen wie z.B. Linien zu suchen. Die Linienextraktion ist ein Teilgebiet der elektronischen Bild- und Messdatenverarbeitung und ist besonders im Bereich der Robotik von großer Bedeutung. Als Anwendungsbeispiele sind u.a. Fahrspur-Erkennung, Positionsbestimmung oder Kontur-Erkennung von Werkstücken bei Industrie-Robotern zu nennen.

Es gibt diverse Verfahren zur Linien-Erkennung, von welchen drei in diesem Abschnitt vorgestellt werden sollen.

5.2.1 Hough-Transformation

Die *Hough-Transformation*, benannt nach Paul Hough (1962) ist eine Technik zur Merkmalserkennung aus Bildern. Die allgemeine Idee dahinter ist das Evaluieren vorhandener Daten im sogenannten *Parameterraum*. Die Parameter der mathematischen Funktion, welche die gesuchte Kontur beschreibt (z.B. Linien, Kreise, Ellipsen) stellen die Achsen des Koordinatensystems im Parameterraum dar.

Hesse’sche Normalform

Eine Gerade kann durch seinen Normalenvektor eindeutig beschrieben werden. Nach der *Hesse’sche Normalform* für Geradengleichungen, der mathematischen Grundlage zur Extraktion von Linien

$$r = x \cdot cos(\phi) + y \cdot sin(\phi) \tag{5.1}$$

setzt sich der Parameterraum für Geraden aus den beiden Achsen für (r, ϕ) zusammen. Abb. 5.3 zeigt den Normalenvektor einer Geraden im karthesischen Koordinatensystem und die Darstellung seiner Polarkoordinaten im *Houghraum* (Parameterraum). Eine Gerade wird als Punkt im Houghraum dargestellt.

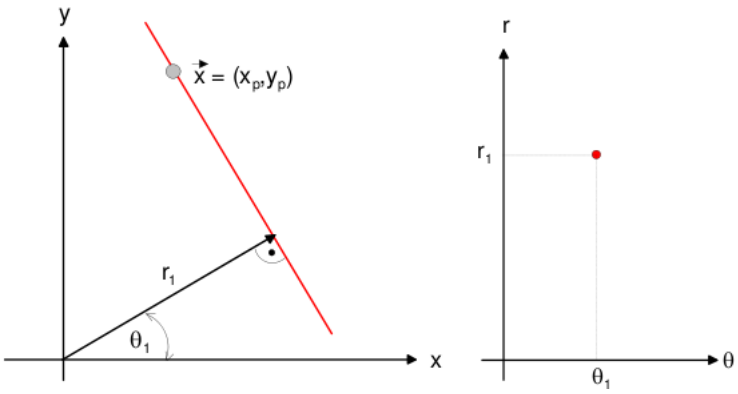


Abbildung 5.3: *Darstellung einer Gerade im karthesischen Koordinatensystem (links) und im Parameterraum (rechts) (Bildquelle: Meisel (WS 2010/11))*

Für die Menge aller durch einen Punkt $\vec{x_1}$ gehenden Geraden gilt

$$r = x_1 \cdot cos(\phi) + y_1 \cdot sin(\phi)$$

mit $\phi \in \left[\phi_{max} - \frac{\pi}{2}, \phi_{max} + \frac{\pi}{2}\right]$. Sie bildet im Houghraum eine nach unten geöffnete Parabel (Abb. 5.4), dessen Maximum den Polarkoordinaten des Ortsvektors $\vec{x_1}$ entspricht:

$$r_{max} = \sqrt{x_1^2 + y_1^2}$$

$$\phi_{max} = arctan\left(\frac{y_1}{x_1}\right)$$

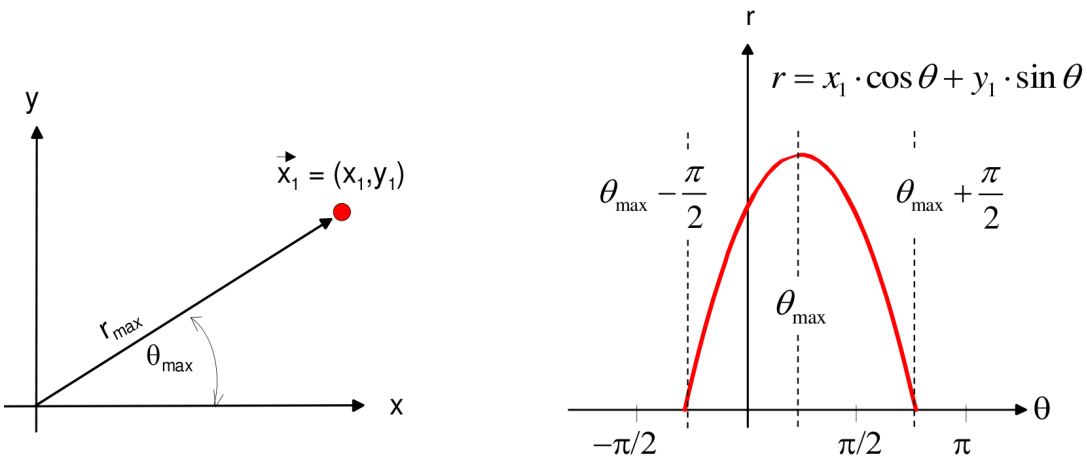


Abbildung 5.4: *Beziehung des Ortsvektors eines Punktes im karthesischen Koordinatensystem (links) zum Maximum seiner im Houghraum gebildeten Parabel (rechts) (Bildquelle: Meisel (WS 2010/11)).*

Um Festzustellen, ob n Punkte im karthesischen Koordinatensystem auf einer Geraden liegen, führt man nun die *Hough-Transformation* durch. Dazu wird für jeden Punkt P_i eine Parabel erzeugt. Schnittpunkte dieser Parabeln deuten auf Kollinearität der entsprechenden Punkte hin.

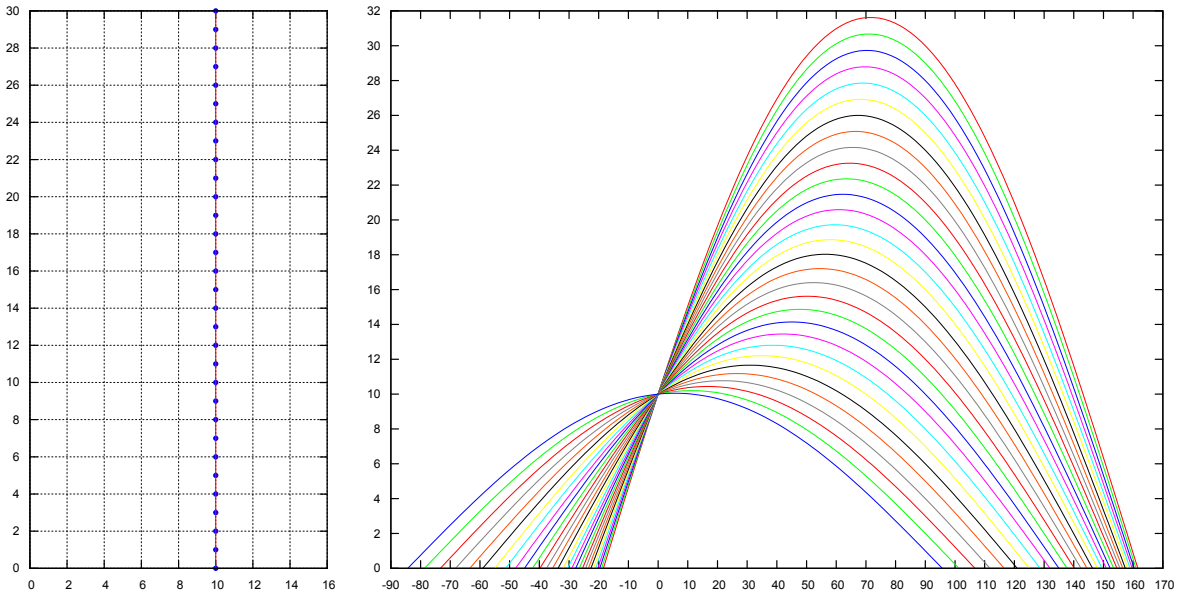


Abbildung 5.5: *Kollineare Punkte im Houghraum*

Abb. 5.5 zeigt 30 auf einer Geraden liegende Punkte (links). Der Schnittpunkt der Parabeln an $r = 10$ und $\phi = 0$ beschreibt den Normalenvektor zur gesuchten Gerade.

Hough-Algorithmus

Ein kontinuierlicher Hough-Raum ist im Computer nicht realisierbar, weshalb er zu dem sog. *Akkumulator-Raum* diskretisiert wird. Der Akkumulator-Raum kann als zweidimensionales Array $(|r| \times |\phi|)$ implementiert werden. Typische Diskretisierungen sind

$$\Delta r = 1 \quad \text{und} \quad \Delta\phi = 2.5^\circ$$

Der Hough-Algorithmus zur Erkennung von Linien sieht wie folgt aus:

1. Berechne für alle n Messpunkte Geraden im Abstand von $\Delta\phi$.
2. Inkrementiere die entsprechenden Stellen im Akkumulator-Raum.

Im Anschluss kann man den Akkumulator-Raum nach lokalen Maxima absuchen, die Indizes entsprechen den HNF-Parametern der gefundenen Geraden.

Anhand des Pseudocodes einer möglichen Implementierung lässt sich die Komplexität des Verfahrens als $O(ns_\phi)$ bestimmen, wobei s_ϕ die Anzahl der Geraden ist, welche durch jeden einzelnen Messpunkt P_i gelegt wird:

```
for i := 1 to n do
    φ_max := arctan(y_i / x_i)
    for φ := φ_max − π/2 to φ := φ_max + π/2 step Δφ do
        r := round(x_i*cos(φ)+y_i*sin(φ))
        A[r][φ]++
    end for
end for
```

Probleme

Neben der relativ hohen Rechenintensität des Verfahrens (optimierte Versionen gibt es in großer Zahl, z.B. FHT[1]) kann die Verteilung der Eingangsdaten zu Problemen führen. Breite Kanten bspw. erzeugen im Akkumulator-Raum eher “Hochplateaus” statt klare Maxima, sodass man ein Bündel dicht aneinanderliegender Geraden erhält. In diesem Fall ist eine Anpassung der Quantisierungsstufe an die Charakteristik der Eingangsdaten notwendig. Alternativ können solche Geradenbüschel auch nach Durchlauf der Hough-Transformation durch Mittelwertbildung zu einer Gerade zusammengefasst werden.

5.2.2 Ausgleichsgerade

Man kann die Aufgabe, Linien aus einer Menge von Messpunkten zu extrahieren, auch als lineares Ausgleichsproblem betrachten. Ein lineares Ausgleichsproblem liegt vor, wenn ein funktionaler Zusammenhang (Geradengleichung) zwar existiert, jedoch die für die Funktion benötigten Parameter fehlen. Stattdessen liegt eine mehr oder weniger große Menge an Messdaten vor. Es gilt nun die Parameter der Geradengleichung

$$f(x) = mx + b$$

so zu bestimmen, dass die Gerade optimal in der Punktewolke liegt (vgl. Abb. 5.6)

[1] Fast Hough-Transform

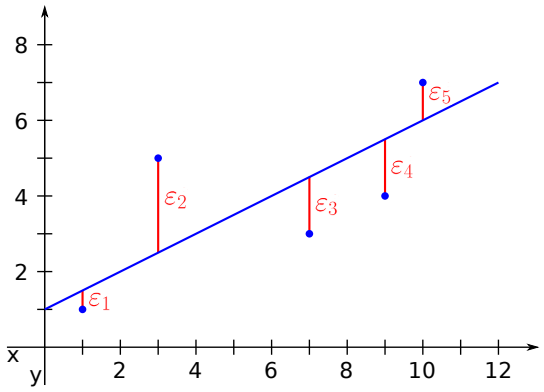


Abbildung 5.6: *Ausgleichsgerade*

Gauß'scher Ansatz

Ein Ansatz zur Bestimmung der optimalen Gerade lässt sich von der Gauss'schen Elimination zur Lösung linearer Gleichungssysteme ableiten. Um die Gerade aus Abb. 5.6 zu erhalten sind die Parameter m und b der Geradengleichung so zu bestimmen, dass die Summe der Fehlerquadrate minimiert wird:

$$\sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (mx_i + b - y_i)^2 \rightarrow minimal \tag{5.2}$$

Um das zu Erreichen kann man Gleichung 5.2 partiell nach m und b ableiten, zu Null setzen und die Unbekannten bestimmen. Etwas eleganter geht es über die transponierte Koeffizientenmatrix. Dazu sei zunächst folgendes LGS zu betrachten:

$$\begin{aligned} mx_1 + b &= y_1 + \varepsilon_1 \\ mx_2 + b &= y_2 + \varepsilon_2 \\ \ldots &= \ldots \\ mx_n + b &= y_n + \varepsilon_n \end{aligned}$$

Der symbolische Fehlervektor $\vec{\varepsilon}$ dient der Vermeidung von Widersprüchen im LGS und wird im weiteren Verlauf nicht gebraucht. Wird er weggelassen, kommt es zu einer Matritzengleichung der allgemeinen Form

$$\underline{A} \cdot \vec{\xi} = \vec{L}$$

mit $\underline{A}$ als Koeffizientenmatrix, dem Unbekanntenvektor $\vec{\xi}$ und dem Lösungsvektor $\vec{L}$:

$$\begin{pmatrix} x_{11} & 1 \\ x_{21} & 1 \\ \dots & \\ x_{n1} & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

Um das Gleichungssystem in eine lösbare Form zu bringen werden beide Seiten mit der transponierten Koeffizientenmatrix $\underline{A}^T$ multipliziert (vgl. Meisel (WS 2010/11)):

$$\underline{A}^T \underline{A} \cdot \vec{\xi} = \underline{A}^T \vec{L} \tag{5.3}$$

Im allgemeinen Fall für n Messpunkte gilt also:

$$\begin{pmatrix} x_{11}^T & x_{12}^T & \dots & x_{1n}^T \\ 1 & 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} x_{11} & 1 \\ x_{21} & 1 \\ \dots & \\ x_{n1} & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} x_{11}^T & x_{12}^T & \dots & x_{1n}^T \\ 1 & 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

$$\begin{pmatrix} \sum x_{1j}^T x_{i1} & \sum x_{i1} \\ \sum x_{i1} & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} \sum x_{1j}^T y_i \\ \sum y_i \end{pmatrix} \tag{5.4}$$

Jetzt handelt es sich wieder um ein gewöhnliches LGS mit zwei Unbekannten, welche per Gauß'scher Elimination oder per Determinantenverfahren (Cramer'sche Regel) eindeutig bestimmt werden können.

Kovarianz

Ein anderer Weg zur Berechnung einer Ausgleichsgerade kommt in Zweigle (WS 2010/11) in einem Algorithmus zur Linien-Extraktion zum Einsatz. Es werden die Parameter der Geradengleichung in Hesse'scher Normalform gesucht, die folgenden Fehler minimiert:

$$E_{fit} = \sum_{i=1}^{n} (x_i \cdot cos(\phi) + y_i \cdot sin(\phi) - r)^2$$

Das Verfahren beruht auf Berechnung der Kovarianz vorliegender Messdaten. Die unbekannten Parameter r und ϕ können bestimmt werden als:

$$\phi = \frac{1}{2} arctan\left(\frac{-2S_{xy}}{S_{y^2} - S_{x^2}}\right) \tag{5.5}$$

$$r = \overline{x} \cdot cos(\phi) + \overline{y} \cdot sin(\phi) \tag{5.6}$$

mit

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

und

$$S_{x^2} = \sum_{i=1}^{n} (x_i - \overline{x})^2 \qquad S_{y^2} = \sum_{i=1}^{n} (y_i - \overline{y})^2$$

$$S_{xy} = \sum_{i=1}^{n} (x_i - \overline{x})\,(y_i - \overline{y}_i)$$

Die Standardabweichung σ^2 als Gütemaß der durch die Messpunkte gebildeten Geraden berechnet sich zu

$$\sigma^2 = \frac{1}{2n}\left(S_{x^2} + S_{y^2} - \sqrt{4S_{xy}^2 + \left(S_{y^2} - S_{x^2}\right)^2}\right) \tag{5.7}$$

Algorithmus

Nachdem zwei mögliche Verfahren zur allgemeinen Berechnung einer Ausgleichsgeraden durch eine gegebene Punktemenge vorgestellt wurden, folgt nun ein auf Ausgleichsrechnung basierender Linienerkennungs-Algorithmus zur Anwendung in der Messdaten-Vorverarbeitung. Folgender Pseudocode zeigt den wesentlichen Ablauf der Hauptroutine, sowie der rekursiven Funktion `split` zum Partitionieren der Punktemenge (vgl. Zweigle (WS 2010/11)):

```
l := empty
start := 0
for i := 1 to n-1 do
    p1 := S[i-1]
    p2 := S[i]
        if distance(p1, p2) > MAX_DISTANCE then
            l := l ∪ split(S, start, i-1)
            start := i
        end if
end for
l := l ∪ split(S, start, n-1)
return l
```

In der Hauptroutine wird eine erste grobe Aufteilung der Messdaten vorgenommen. Auf diese Weise können Messpunkte, die von relativ weit voneinander entfernten Objekten stammen getrennt voneinander an die eigentliche Linienerkennungsfunktion übergeben werden.

Der Parameter `MAX_DISTANCE` bestimmt den Schwellwert, ab welcher Entfernung Messpunkte nicht mehr zusammengehören. Sind zwei Punkte p_{i-1} und p_i weiter voneinander entfernt als `MAX_DISTANCE`, so wird die Punktemenge an dieser Stelle partitioniert und alle Punkte bis einschliesslich p_{i-1} zu einer Teilmenge zusammengefasst.

```
function split(S, start, end)
l := empty
line := make_line(S, start, end)
    if numpoints(line) ≥ MIN_POINTS_ON_LINE then
        if sigma(line) < MAX_SIGMA then
            l := l ∪ {line}
        else
            p_start := S[start]
            p_end := S[end]
            i_split := start
            d := 0
            for i:=start+1 to end-1 do
                p := S[i]
                if distance_to_line(p, p_start, p_end) > d then
                    i_split := i
                    d := distance_to_line(p, p_start, p_end)
                end if
            end for
        l := l ∪ split(S, start, i_split)
        l := l ∪ split(S, i_split, end)
    return l
end function
```

Die `split`-Funktion wird rekursiv aufgerufen und führt auf den übergebenen Messdaten der Reihe nach folgende Schritte aus:

1. Berechne Ausgleichsgerade (`make_line`). Hierzu kann eines der oben erwähnten Verfahren eingesetzt werden. Da das Gauß-Verfahren (vgl. 5.2.2) keine direkte Information über die Standardabweichung bietet, wird der zweite Ansatz (vgl. 5.2.2) verwendet.

2. Untersuche die soeben erzeugte Ausgleichsgerade und verwerfe sie, wenn sie aus weniger als `MIN_POINTS_ON_LINE` berechnet wurde, sonst

   a) speichere die Gerade ab, wenn die Standardabweichung unter `MAX_SIGMA` liegt, sonst

   b) suche den Punkt aus der Ausgleichsgeraden, der die größte Entfernung zur Geraden hat, teile die Punktemenge an der Stelle auf und rufe `split` erneut auf für jede der beiden Teilmengen.

Die beiden Parameter `MIN_POINTS_ON_LINE` und `MAX_SIGMA` steuern die Menge und Qualität der Geraden. Je größer der Wert für `MIN_POINTS_ON_LINE` umso weniger Geraden werden erzeugt. Mit größerem `MAX_SIGMA` steigt die Zahl der Geraden — gleichzeitig nimmt jedoch ihre Genauigkeit ab.

Der Algorithmus gehört zu den *divide&conquer*-Algorithmen. Die Zeitkomplexität lässt sich wie bei anderen Algorithmen dieser Klasse im *average case* mit $O(n \log n)$ und im *worst case* mit $O(n^2)$ abschätzen.

5.2.3 Medianfilter

Die oben vorgestellten Ansätze, Geraden aus Messdaten zu extrahieren sind anfällig gegen sogenannte Ausreißer, das sind einzelne über die Weltkarte verstreute Messwerte, die durch Sensorrauschen oder kleine Objekte im Raum hervorgerufen werden können.

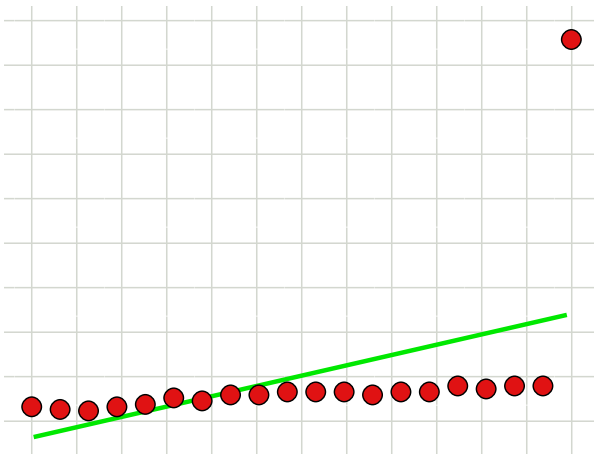


Abbildung 5.7: *Ein Ausreißer zieht Ausgleichsgerade nach oben (Bildquelle:Wikipedia (2011))*

Es macht daher Sinn, vor der Linienextraktion eine Medianfilterung durchzuführen, um solche Ausreißer zu eliminieren.

Ein Medianfilter für LRF[2]-Messdaten formuliert sich im Prinzip nicht anders als ein Medianfilter in der Verarbeitung von Bilddaten oder anderen Signalen:

1. Definiere eine ungerade Maskengröße m
2. Setze den Entfernungswert (d.h. den gemessenen Wert an Winkel ϕ_i) für jeden Messpunkt p_i gleich dem Median aus den $\frac{m}{2}$ Messpunkten vor und nach ihm, einschliesslich p_i selbst

Auf diese Weise können einzelne Ausreisser zuverlässig und mit relativ wenig Aufwand unschädlich gemacht werden. Das Verfahren hat eine Komplexität von $O(mn)$, wobei die Maskengröße m meist vernachlässigbar klein gewählt wird und somit nur die Zahl der Messpunkte wirklich ins Gewicht fällt: $O(n)$. Ähnlich wie beim Reduktionsfilter (vgl. 5.1) hat auch der Medianfilter den Nachteil, Ecken abzurunden.

5.3 Reduzierung der Schleifendurchläufe

Nachdem einige Themen bezüglich der Messdatenvorverarbeitung besprochen wurden, soll in diesem Abschnitt auf die andere Stellgröße zur Laufzeit-Optimierung eingegangen werden: die Anzahl der Transformationen und Abstandsberechnungen.

5.3.1 Mittlere Abstände im Suchraum

Wie in Abb. 4.5 bereits zu beobachten war, haben die mittleren Abstände im durchsuchten Fitting-Bereich einen talförmigen Verlauf. Deutlicher wird dieses Verhalten, wenn man die innerste Schleife des Algorithmus' (vgl. 4.4.1) betrachtet, in der die y-Transformation berechnet wird:

[2] Laser Range Finder

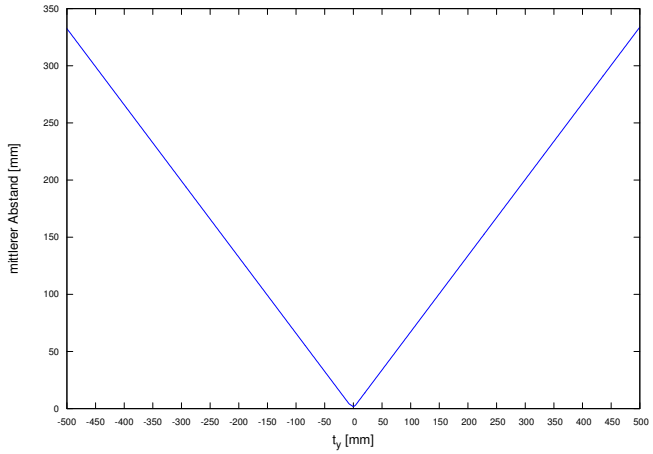


Abbildung 5.8: *Querschnitt der 3D-Oberfläche aus 4.5 an* $t_x = 500$

Man kann diese Eigenschaft ausnutzen und aus der `for`-Schleife springen wenn man feststellt, dass der Graph wieder ansteigt. Auf diese Weise lassen sich unnötige Berechnungen einsparen.

Umschalten der Suchrichtung

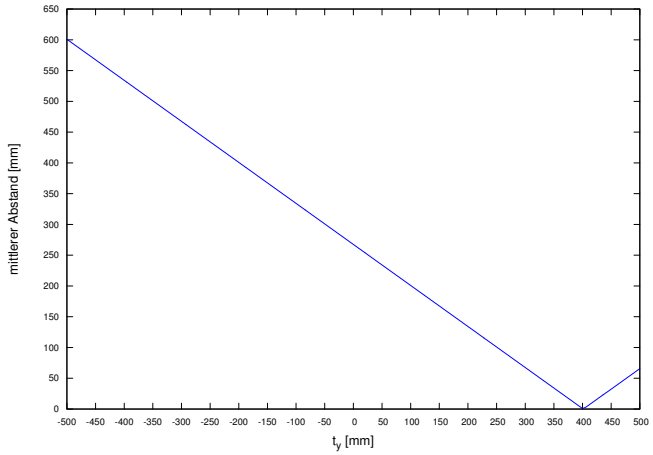


Abbildung 5.9: *Verlauf der Abstände bei 400mm y-Translation zum Referenzpolygon*

Hat man aufgrund eines groben Schätzverfahrens für die geschätzte Position einen relativ großen Bereich zu durchsuchen, kann es Sinn machen das Vorzeichen der letzten gefundenen Transformation zu betrachten. Hintergrund für diese Überlegung ist, dass Fahrzeuge in der Regel keine plötzlichen 180°-Wendungen fahren. Abb. 5.9 zeigt einen Fall in dem

sich das MS seit seiner letzten bekannten Position um $400mm$ in positiver y-Richtung bewegt hat. Es ist unwahrscheinlich, dass das MS plötzlich die y-Richtung umkehrt und nach "unten" fährt.

Würde man den y-Raum von positiv nach negativ durchsuchen statt andersherum, wäre in dieser Situation eine Reduzierung der Schleifendurchgänge auf nur ca. 10% des ursprünglichen Wertes möglich (im Vergleich zum vollen Suchgang über das gesamte Intervall $[-500, 500]mm$).

5.3.2 Eingrenzung des Suchraumes

Die oben verwendeten Suchintervalle für die x/y-Translation wurden willkürlich gewählt. Ein Suchbereich von $1000 \times 1000mm$ wäre für die Praxis nicht sinnvoll, da er bei einer realistisch kleinen Schrittweite zu viel Rechenzeit kosten würde (vgl. Gleichung 4.6). Es gilt daher, die neue Pose im Vorfeld ausreichend genau abzuschätzen, um darauf aufbauend mit dem scanbasierten Algorithmus gute Werte für Auflösung und Aktualisierungsrate zu erzielen.

worst case Abschätzung

Einen ersten groben Anhaltspunkt, wie die Intervallgrößen gewählt werden sollten, bieten Fahrtgeschwindigkeit und durchschnittliche Rechendauer pro Positionsaktualisierung. Bewegt sich das mobile System bspw. mit $1ms^{-1}$ fort und braucht der Algorithmus samt Laserscan $120ms$, so muss es sich in einem Radius von $120mm$ um die letzte Position befinden. Es muss also ein recht großer Bereich von $240 \times 240mm$ für die Translation durchsucht werden. Über die geschätzte Orientierung des MS kann dieser Ansatz keine brauchbare Aussage machen. Aus diesen Gründen ist die Übernahme der letzten bekannten Position als neue geschätzte Position nur bei langsamer Fahrt oder geringer Auflösung brauchbar.

Einfache Koppelnavigation

Eine bessere Eingrenzung des Suchbereiches kann durch eine vereinfachte Koppelnavigation erreicht werden. Als Eingangsgröße wird lediglich die Geschwindigkeit des Fahrzeugs benötigt. Die Fahrtrichtung kann intern anhand der letzten Positionen berechnet werden — der Lenkwinkel wird hierfür nicht betrachtet, da analog zur Messdatenkorrektur in Abschnitt 3.3.2 von einer geradlinigen Bewegung ausgegangen wird. Der durch diese Pauschalisierung entstehende maximale Translations-Fehler berechnet sich nach Gleichung 3.3.

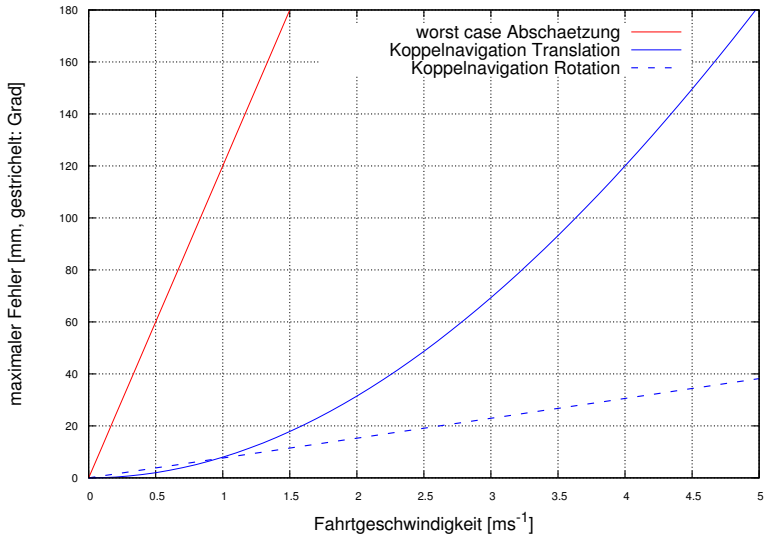


Abbildung 5.10: *Maximaler Fehler in Abhängigkeit der Fahrtgeschwindigkeit bei worst case Schätzung und vereinfachter Koppelnavigation*

Mit der einfachen Kopplung liefert das Verfahren auf unserem Testsystem[3] bereits recht genaue Positionswerte ($1mm$ Auflösung Translation und $0.5°$ Auflösung Rotation) bei $1ms^{-1}$ Fahrt und ca. $30ms$ Algorithmenlaufzeit.

Odometrie

Höhere Fahrtgeschwindigkeiten bei gleichbleibender oder besserer Positionsauflösung sind möglich, wenn man zusätzlich noch Größen wie Lenkwinkel und Radstand der mobilen Plattform miteinbezieht.

Abb. 5.11 zeigt das zweispurige Fahrwerksmodell mit Vorderachslenkung und starrer Hinterachse. Die Eingangsgrößen für die Odometrieberechnung sind rot markiert:

- der Lenkwinkel δ wird mittels Lenkwinkelsensor erfasst
- der Radstand l liegt als feste Größe vor
- die gefahrene Strecke s innerhalb eines Zeitabschnitts Δt kann aufgrund fehlender Radsensoren nur indirekt anhand von Motordrehzahl, Radumfang, Gangstellung und Getriebeübersetzung berechnet werden. Alternativ kann auch einfach durch Messung von Rundenzeiten versucht werden, eine Funktion für die resultierende Geschwindigkeit in Abhängigkeit von Gang und Motordrehzahl anzunähern.

[3] Intel® Core™ 2Duo L7100 @1.2 GHz

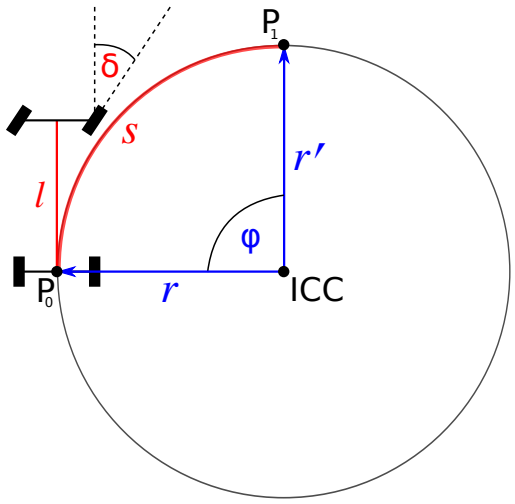


Abbildung 5.11: *Abschätzung der neuen Pose mittels Odometrie*

Ein Fahzeug, welches eine Strecke s mit konstantem Lenkwinkel δ entlangfährt, bewegt sich während dieser Fahrt auf einer Kreisbahn mit dem Radius r. Sieht man den r als Ortsvektor zum Bezugspunkt des Fahrzeugs P_0, so lässt sich die Berechnung des Endpunktes P_1 als Vektor-Rotation um den sog. *instantaneous center of curvation (ICC)* formulieren (vgl. Zoebel (2003)):

$$P_1 = \begin{pmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{pmatrix} \begin{pmatrix} x_0 - ICC_x \\ y_0 - ICC_y \end{pmatrix} + \begin{pmatrix} ICC_x \\ ICC_y \end{pmatrix} \tag{5.8}$$

Für den Rotationswinkel ϕ gilt im Bogenmaß:

$$\phi = \frac{s \cdot tan(\delta)}{l} \text{rad} \tag{5.9}$$

Weil r auf einer Geraden durch die Hinterachse der mobilen Plattform liegt, befindet sich der Fahrkreismittelpunkt orthogonal zur Längsachse des Fahrzeugs (rechts bei negativen Lenkwinkeln und links bei positiven). Das bedeutet für die geschätzte Orientierung:

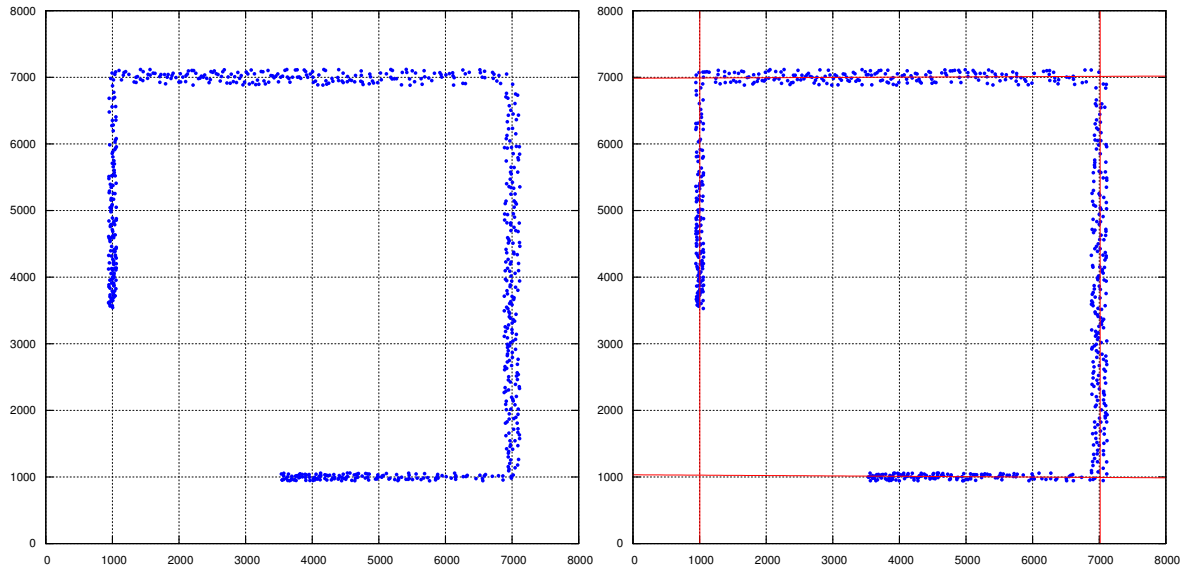
$$\text{Orientierung}_1 = \begin{cases} arctan\left(\frac{y_1}{x_1}\right) - \frac{\pi}{2}, & \text{wenn } \delta < 0 \\ arctan\left(\frac{y_1}{x_1}\right) + \frac{\pi}{2}, & \text{wenn } \delta > 0 \end{cases} \tag{5.10}$$

5.4 Kartenerstellung

In Abschnitt 4.4 wurde zunächst noch von einer *a priori*-Karte ausgegangen, d.h. das Referenzpolygon existiert bereits zur Initialisierungszeit des Algorithmus'. Die Frage wo dieses Polygon herkommt, ist bislang noch nicht geklärt worden. Eine Methode stellt natürlich das manuelle Ausmessen der Einsatzumgebung dar, was jedoch relativ anstrengend werden kann und wenig flexibel ist.

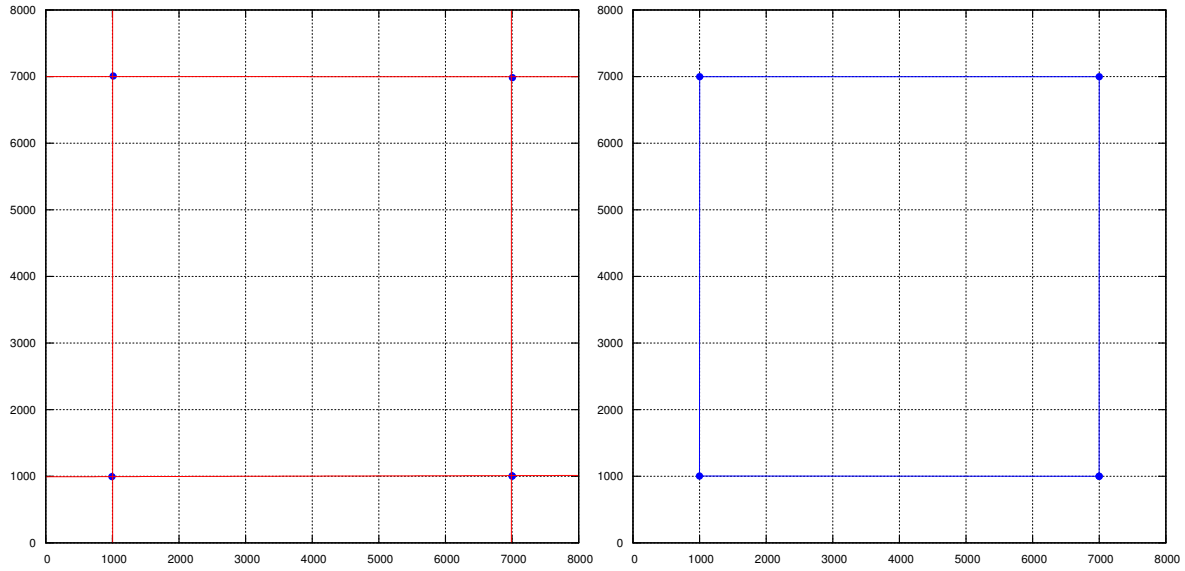
In diesem Abschnitt soll gezeigt werden, dass es mit den bisher vorgestellten Mitteln relativ einfach ist, das Referenzpolygon vom autonomen System zur Initialisierungszeit selbst erstellen zu lassen. Das bietet neben besserer Flexiblität und leichterer Handhabung den Vorteil, dass das Ausmessen des Referenzpolygons und später im laufenden Betrieb die Distanzmessungen mit den gleichen Werkzeugen durchgeführt werden. Auf diese Weise werden systematische Unterschiede zwischen den Messungen umgangen.

Abb. 5.12 und 5.13 zeigen eine simulierte Kartenerstellung: aus einer günstigen Pose wird ein $240°$-Scan durchgeführt und anschließend ein Polygon daraus berechnet. Das Referenzpolygon für die spätere Positionsbestimmung ist das Mittel aus n Einzelpolygonen, um Abweichungen durch Messungenauigkeiten auszugleichen. Die für die Initialisierung benötigte Zeit beträgt aufgrund der maximalen Scandrate von 10Hz ca. $n \cdot 100ms$. Dies spielt für die Aktualisierungsrate der Positionsbestimmung zwar keine Rolle, sollte jedoch bei der Parametrisierung der Software beachtet werden.



(a) Rohscan (b) Geraden-Erkennung

Abbildung 5.12: *Für die Berechnung eines Polygons wird zunächst ein simulierter 240°-Scan durchgeführt und anschließend der Linienerkennungs-Alorithmus aus Abschnitt 5.2.2 angewendet.*



(a) Schnittpunksberechnung (b) Polygonbildung

Abbildung 5.13: *Im nächsten Schritt werden die Schnittpunkte der aus dem Scan gewonnenen Ausgleichsgeraden berechnet und als Eckpunkte des gesuchten Polygons definiert.*

6 Design und Realisierung

Dieses Kapitel dokumentiert auf den bisher erarbeiteten Konzepten aufbauend die Entwicklung einer MinGW[1]-basierten Laufzeitumgebung zur Positionsbestimmung mobiler Systeme mit Hilfe eines Laserscanners.

6.1 Architektur

Abb. 6.1 zeigt die Toplevel-Ansicht des mobilen Systems mit den Hauptkomponenten. Die Mobile Plattform *Peterbilt 359* wird über ein ARM-Board der Firma NXP gesteuert, welches als HID[2] über eine USB-Schnittstelle mit dem Hauptrechner kommuniziert.

Das Kontrollprogramm auf dem Hauptrechner empfängt Sensordaten (z.B. Lenkwinkel, Motordrehzahl) vom ARM-Board und sendet seinerseits Befehle zum Beschleunigen, Bremsen und Lenken. Als weitere Aufgabe übernimmt das Kontrollprogramm die Streckenplanung und Fahrtkontrolle. Dies geschieht über die Berechnung von Beziér-Kurven mit Hilfe einer Laufzeitumgebung zur Laser-gestützten Positionsbestimmung (`libposition.dll`).

Das Positionsbestimmungs-Subsystem aktualisiert mit einer Taktrate von ca. 8Hz laufend die Pose der mobilen Plattform. Sie verwendet dazu Odometrie-Informationen vom Kontrollprogramm und Distanzmesswerte aus Laserscans. Die Kommunikation zum Laserscanner geschieht über das SCIP-Protokoll (Version 2.0). Eine Implementierung dieses Protokolls in der Form statischer C-Bibliotheken steht auf der Homepage des Herstellers zur Verfügung (vgl. Hokuyo).

Das Modul, um welches es in dieser Arbeit geht ist in Abb. 6.1 grün markiert. Auf die anderen Module wird bis auf die Aufführung einiger Funktionen aus dem URG Library Interface nicht weiter eingegangen.

[1] Minimalist GNU for Windows
[2] Human Interface Device

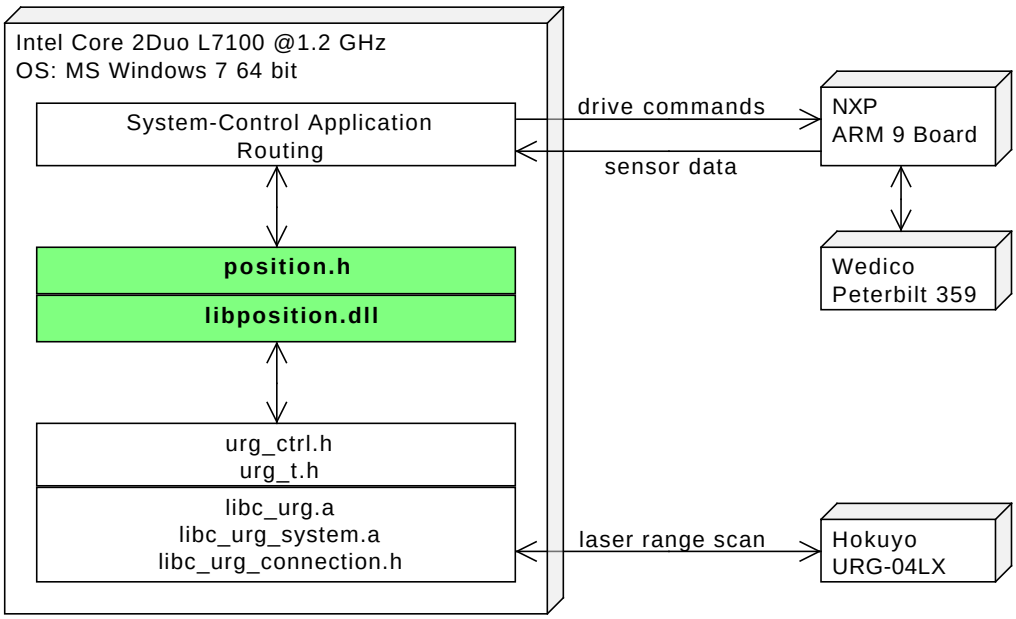


Abbildung 6.1: *Implementierung der Laufzeitumgebung als MinGW Windows-DLL*

Innerer Aufbau

Der innere Aufbau der DLL ist in Abb. 6.2 dargestellt. Sie besteht aus vier Hauptkomponenten:

`position:` Dies ist die Hauptkomponente und zugleich die Schnittstelle nach Aussen. Eine Anwendung (z.B. das Kontrollprogramm) die über dieses Interface die Laufzeitumgebung in Betrieb nimmt, übergibt bei der Initialisierung einen Zeiger auf eine Funktion mit der Signatur
`void (*cbfunc)(int xpos, int ypos, double phi)`. Sobald die Laufzeitumgebung gestartet wurde, beginnt sie, laufend die aktuelle Position des MS zu bestimmen und ruft nach jeder Positionsaktualisierung diese Funktion mit den ermittelten Posen-Informationen als Argumente auf.

`scan` kapselt den Zugriff auf den Laserscanner URG-04LX über die `libc_urg_*`-Bibliotheken.

`scansim` simuliert die in der technischen Spezifikation (vgl. Tabelle 3.1) angegebenen Eigenschaften des Laserscanners.

util: Hier sind alle Hilfsfunktionen und Datenstrukturen definiert, welche bei der Positionsbestimmung zum Einsatz kommen (z.B. Funktionen zur Transformation von Punktemengen, Merkmalsextraktion, Odometrieberechnung sowie zwei Filter zur Messdatenvorverarbeitung).

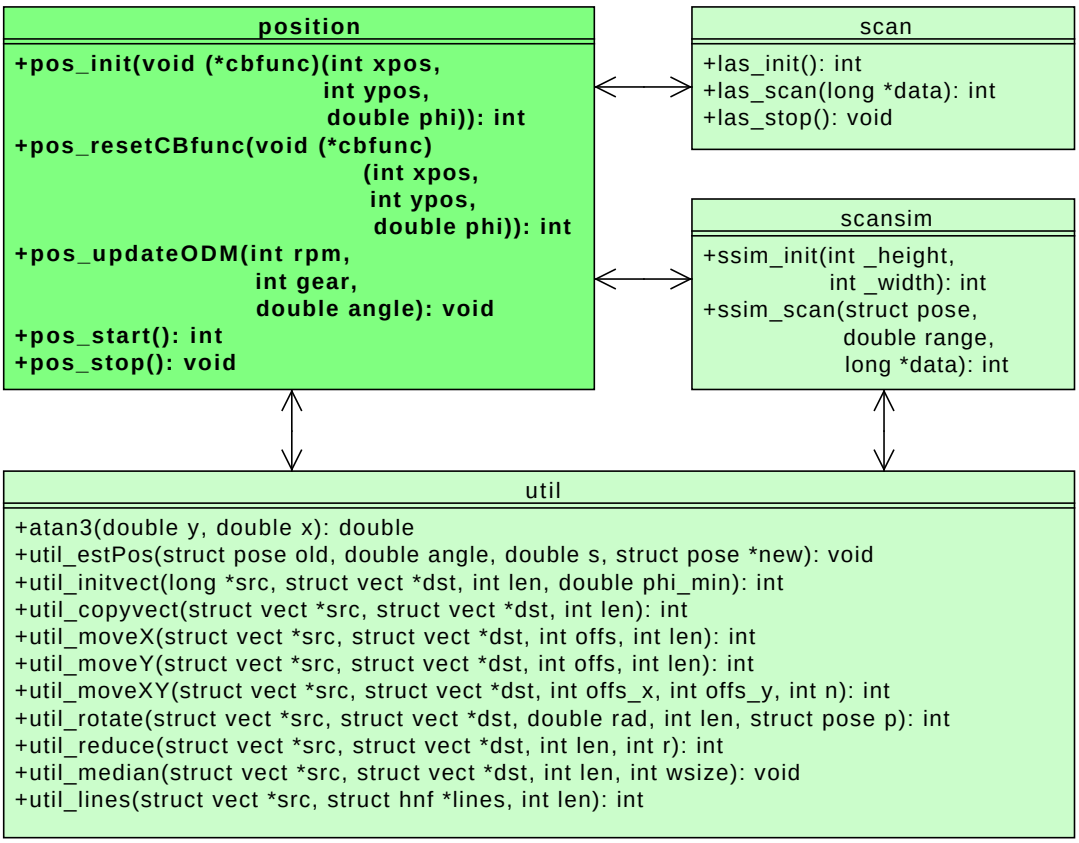


Abbildung 6.2: *Innerer Aufbau der Positionsbestimmungs-Software*

6.2 Ablauf

In diesem Abschnitt wird zunächst geschildert, wie die Laufzeitumgebung zur Positionsbestimmung von einer Anwendersoftware in Betrieb genommen werden kann. Das Interface zur DLL ist im Anhang gelistet (vgl. Listing 1).

Zunächst muss die Laufzeitumgebung initialisiert werden, dies geschieht über den Aufruf der Funktion `pos_init(...)`. Als Übergabeparameter muss der Aufrufende Prozess einen Zeiger auf eine Funktion angeben, welche bei jeder Positionsaktualisierung aufgerufen wird.

Diese Callback-Funktion kann später jederzeit über `pos_resetCBfunc(...)` durch eine andere ersetzt werden, ohne die Laufzeitumgebung stoppen zu müssen. Während der Initialisierungsphase werden ausserdem noch folgende Schritte ausgeführt:

- Initialisieren von lokalen Datenstrukturen.
- Vermessen der Einsatzumgebung und Berechnung des Referenz-Polygons.
- Vorbereiten von Transformationsparametern, d.h. sämtliche arithmetischen Terme, die im Positionsbestimmungs-Algorithmus vorkommen so weit wie möglich vorausberechnen und in Arrays speichern.

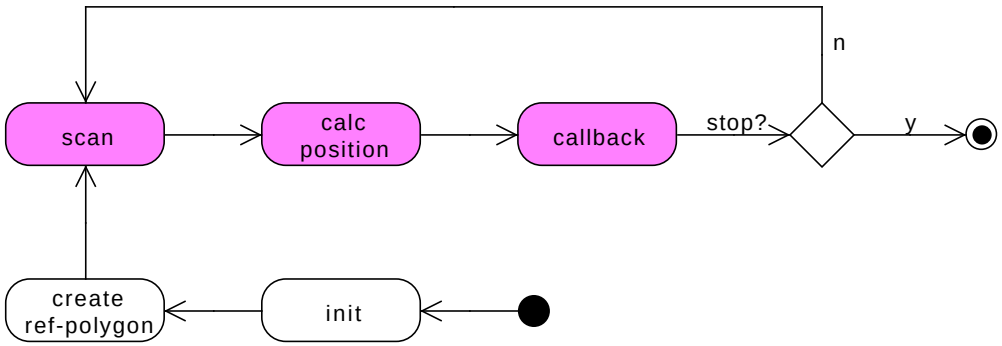


Abbildung 6.3: *Ablaufdiagramm des Positionsbestimmungs-Systems*

Nach erfolgreicher Initialisierung kann durch den Aufruf von `pos_start()` die Positionsbestimmung gestartet werden. Dazu wird ein Thread gestartet, welcher in einer Endlosschleife laufend Messdaten vom Laserscanner abruft und dessen Pose in der zuvor erstellten Weltkarte aktualisiert. Diese Information wird per Aufruf der registrierten Callback-Funktion an den Anwender-Prozess übergeben.

Änderungen am Bewegungszustand der mobilen Plattform (d.h. nach jedem Lenk- bzw. Beschleunigungsbefehl an das ARM-Board) müssen der Laufzeitumgebung mit Hilfe der Funktion `pos_updateODM(...)` mitgeteilt werden, damit die Berechnung der geschätzten Position mit “korrekten”[3] Odometrie-Daten durchgeführt werden kann.

Dieses wird fortgeführt bis die Anwendung entweder `pos_stop()` aufruft, oder terminiert.

[3] vgl. 2.2.1

Positionsbestimmung

Nachdem ein Überblick über den generellen Ablauf gegeben wurde, soll der Fokus nun auf die Hauptroutine, die `run(...)`-Funktion gerichtet werden. Diese Funktion implementiert den in Abschnitt 4.4 vorgestellten Algorithmus zur Positionsbestimmung durch Überdeckung von Distanzmesswerten mit Polygonen. In dieser konkreten Implementierung wurden die Überlegungen aus Kapitel 3 und 5 umgesetzt.

Listing 6.1 zeigt den Quellcode der Positionsbestimmungs-Schleife:

Listing 6.1: Positionsbestimmungs-Algorithmus

```
/**
 * Positionsbestimmungs-(Endlos-)Schleife
 */
static void *run(void *unused) {

    struct pose pos, pos_g;
    struct vect m[SCANSIZE], m1[SCANSIZE], m2[SCANSIZE], m3[SCANSIZE];
    struct hnf lines[20];
    double dist, last, closest, phi_offset, scan_angle0, deltaT;
    int phi_i, phi = 0, x_i, x = 0, y_i, y = 0, ctr, lctr = 0, i;
    int x_offset, y_offset;
    int phidown = 0, xdown = 0, ydown = 0;
    long t1, t2;
#ifdef CORRECTION
    struct pose d;
    int ds;
    double alpha, v;
#endif

#ifdef SIM
    // simulierte Ausgangsposition
    pos.x = 3;
    pos.y = 10;
    pos.phi = 0.8;
#endif

    // geschaetzte Ausgangsposition
    pos_g.x = 0;
    pos_g.y = 0;
    pos_g.phi = 0;

    while (running) {

        // Timestamp 1
        t1 = timeGetTime();
```

```
        closest = dmax;

        // Messdaten holen
#ifndef SIM
        scansize = las_scan(rawdata);
        scan_angle0 = static_scan_angle0;
#else
        scansize = ssim_scan(pos, SCANRANGE, rawdata);
        scan_angle0 = DEG2RAD(((int)((360-SCANRANGE/2)+pos_g.phi))
           %360);
#endif

        /* Messdaten ins Weltmodell uebertragen und Hilfsgroessen
        berechnen */
        scansize = util_initvect(rawdata, m, scansize, scan_angle0);

        // Messdaten filtern
        scansize = util_lines(m, lines, scansize);
        for(lctr = 0, i = 0; lctr < scansize; lctr++) {
            m[i++] = lines[lctr].rep1;
            m[i++] = lines[lctr].rep2;
        }
        scansize = i;

        // Messdaten auf KO-Ursprung verschieben
        util_rotate(m, m, DEG2RAD(pos_g.phi), scansize, pos_g);
        util_moveXY(m, m, pos_g.x, pos_g.y, scansize);

        // Transformationsraum durchsuchen
        for (phi_i = 0; phi_i <= PHIRANGE; phi_i++) {
            util_rotate(m, m1, phi_array[phi_i], scansize, pos_g);
            for (x_i = 0; x_i <= XRANGE; x_i++) {
                util_moveX(m1, m2, x_array[x_i], scansize);
                last = 0;
                ctr = CTR_THRESHOLD;
                for (y_i = 0; y_i <= YRANGE; y_i++) {
                    util_moveY(m2, m3, y_array[y_i], scansize);
                    // Mittlere Abstaende berechnen
                    dist = calc_dist(m3);
                    // Fitting-Parameter mit geringstem Abstand merken
                    if(dist < closest) {
                        closest = dist;
                        phi = phi_i;
                        x = x_i;
                        y = y_i;
                    }
                    /* wird Abstand groesser ? Wenn ja, raus aus
                    y-Schleife */
```

```
                    if(y_i && dist > last) ctr--;
                    if(!ctr) break;
                    last = dist;
                }
            }
        }

        /* Fitting Parameter wenn noetig umkehren
        (aufsteigend/absteigend) */
        if(phidown) phi = PHIRANGE-phi;
        if(xdown) x = XRANGE-x;
        if(ydown) y = YRANGE-y;

        // x-y-phi Abweichungen raussuchen
        phi_offset = phi_asc[phi];
        x_offset = x_asc[x];
        y_offset = y_asc[y];

        // Position korrigieren
        pos.x = pos_g.x + x_offset;
        pos.y = pos_g.y + y_offset;
        pos.phi = pos_g.phi + phi_offset;

        // Timestamp 2 und Delta t berechnen
        t2 = timeGetTime();
        deltaT = (double) t2-t1;

#ifdef CORRECTION
        // Versatz fuer Messdatenkorrektur ermitteln
        d.x = pos_g.x-pos.x;
        d.y = pos_g.y-pos.y;
        alpha = atan3(d.y, d.x);
        v = sqrt(d.x*d.x+d.y*d.y)/deltaT;
        ds = (100*v*SCANRES_R)/B_2PI;
        dx = (double) round(ds*cos(alpha));
        dy = (double) round(ds*sin(alpha));
#endif

        // geschaetzte Position berechnen
        util_estPos(pos, odm.angle, odm.speed*deltaT, &pos_g);

        /* neue Runde vorbereiten (Durchlaufreihenfolge der Achsen
        dort umkehren wo es Sinn macht) */
        if(phi_offset > 0) {
            phi_array = phi_desc;
            phidown = 1;
        } else {
            phi_array = phi_asc;
```

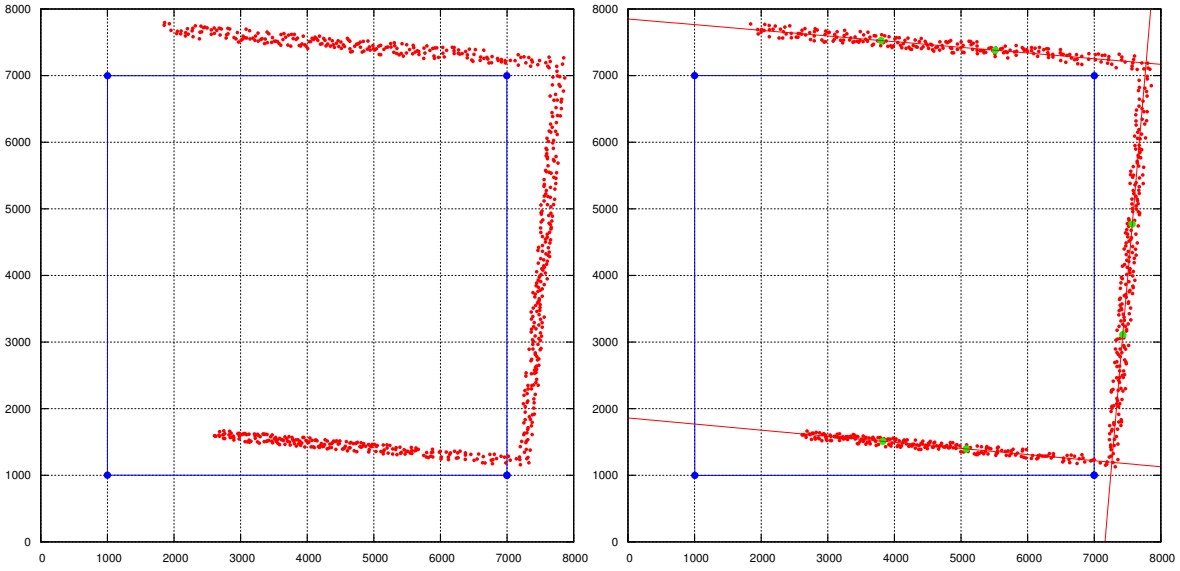
```
            phidown = 0;
        }
        if(x_offset > 0) {
            x_array = x_desc;
            xdown = 1;
        } else {
            x_array = x_asc;
            xdown = 0;
        }
        if(y_offset > 0) {
            y_array = y_desc;
            ydown = 1;
        } else {
            y_array = y_asc;
            ydown = 0;
        }

        // Position und Orientierung an Anwendung uebergeben
        callback(pos.pos_x, pos.pos_y, pos.angle);
    }

        return EXIT_SUCCESS;
}
```

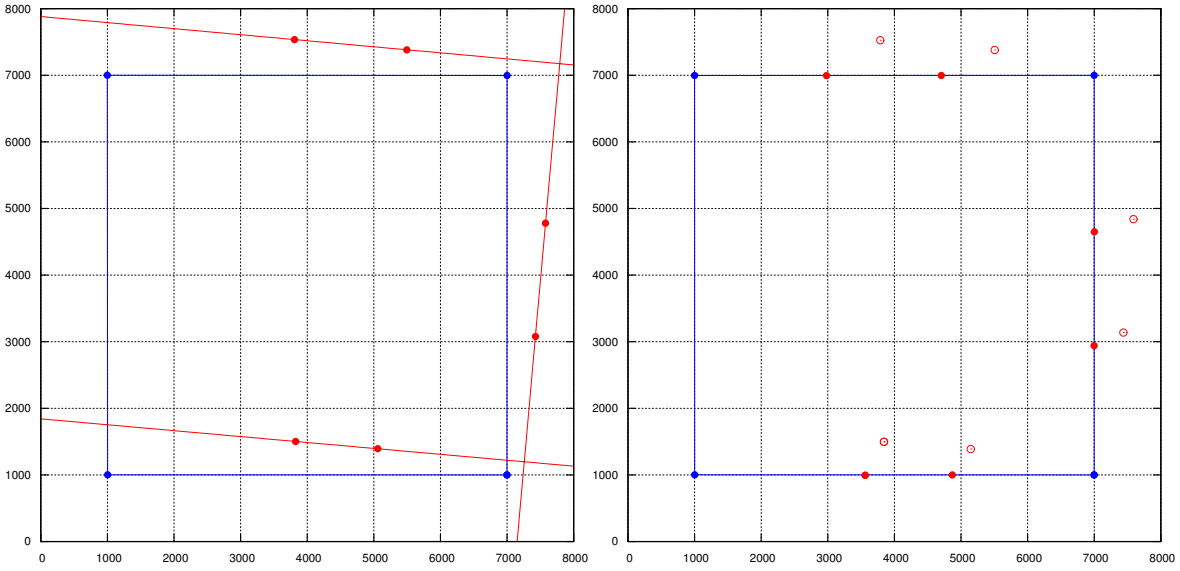
Abb. 6.4 und 6.5 zeigen den Ablauf einer Positionsbestimmungsrunde: Zunächst wird der Linienerkennungs-Algorithmus auf die Punktemenge angewendet. Im nächsten Schritt werden für jede erkannte Gerade jeweils zwei Punkte als Repräsentanten definiert, welche gut in dem jeweiligen Segment von Messpunkten und gleichzeitig auf den Ausgleichsgeraden liegen.

Nachdem das erledigt ist, werden die rohen Distanzmesswerte verworfen und der Positionsbestimmungs-Algorithmus auf deren Repräsentanten angewendet. Auf diese Weise wird die Zahl der Punkte, die in jedem Schleifendurchlauf während der Suche nach der richtigen Transformation verschoben und verdreht werden müssen, ohne Informationsverlust auf einen Bruchteil dessen reduziert was anfangs war.



(a) Rohscan (b) Geraden-Erkennung und Repräsentantenwahl

Abbildung 6.4: *Vorverarbeitung der Distanzmesswerte (rot)*



(a) Ersetzen der Messdaten durch Repräsentanten (b) Bestimmung der Transformation

Abbildung 6.5: *Reduzierung der Zahl zu transformierender Punkte von 682 auf 6*

6.3 Werkzeuge

In diesem Abschnitt werden die wichtigsten Werkzeuge vorgestellt, die bei der Entwicklung des Positionsbestimmungssystems hilfreich waren.

6.3.1 Laserscanner-Simulator

Für die Generierung von Messdaten wurde ein Plugin geschrieben, welches den Laserscanner Hokuyo URG-04LX-UG01 mit den in Tabelle 3.1 aufgeführten technischen Eigenschaften simuliert.

Bei der Initialisierung durch Aufruf der Funktion
`int ssim_init(int _height, int _width)`
werden die Ausmaße des zu simulierenden Raumes übergeben. Zusätzlich können noch weitere Parameter wie Streufaktor und Reichweite konfiguriert werden. Dies geschieht über die Header-Datei `param.h`:

Listing 6.2: Parametrisierung des Laserscanner-Simulators (param.h)

```
#define SCANRES_D       0.3515625   // Winkelschrittweite (Grad)
#define SCANRANGE       240         // Messwinkel (Grad)
#define SCANSIZE        682         // Anzahl Messpunkte
#define SCANDEVIANCE    3           // Messungenauigkeit (Prozent)
#define RMIN            20          // Mindestdistanz (mm)
#define RMAX            5600        // Maximaldistanz (mm)
#define MAPWIDTH        6000        // Raumbreite (mm)
#define MAPHEIGHT       6000        // Raumtiefe (mm)
```

6.3.2 gnuplot

Für die visuelle Darstellung von Messdaten wurde das interaktive kommandozeilenorientierte open source Programm `gnuplot` eingesetzt. Es unterstützt unter anderem die Plattformen Linux, OS/2, MS Windows, OS X und VMS.

Das `gnuplot`-Projekt entstand 1986 um Wissenschaftlern und Studenten es zu ermöglichen, mathematische Funktionen und Datenfelder zu visualisieren. Mit der Zeit wuchs es mehr und mehr aus dieser alleinigen Rolle heraus und wird heutzutage aufgrund seiner skriptfähigen Architektur und der Vielzahl an möglichen Ausgabeformaten (z.B. screen terminals, direkte Drucker-/Plotterausgabe, LaTeX, metafont, `eps, fig, jpeg, pbm, pdf,`

`png, ps, svg, ...`) in diversen anderen Bereichen genutzt. So wird es beispielsweise in `GNU Octave`, einem Programm für numerische Berechnungen, als plot-engine eingesetzt.

Bei der Entwicklung des Positionsbestimmungssystems erwies sich die Skriptfähigkeit von `gnuplot` ebenfalls als sehr hilfreich. Der Arbeitsablauf beim Debuggen ließ sich mit diesem Werkzeug sehr gut automatisieren:

Die x/y-Koordianten der Messpunkte wurden zur Laufzeit in Datendateien geschrieben und die Funktionsgleichungen der Ausgleichsgeraden gemeinsam mit der gewünschten plot-Konfiguration in einer `gnuplot`-Skriptdatei gespeichert. Der eigentliche Aufruf von `gnuplot` erfolgte dann wiederum in einem Bash-Skript nach Durchlauf der Positionsbestimmung. Auf diese Weise stand neben den Debug-Informationen durch `printf`-Statements in der Konsole ebenso augenblicklich eine visuelle Darstellung der Vorgänge zur Verfügung (vgl. Abb. 6.4 und 6.5).

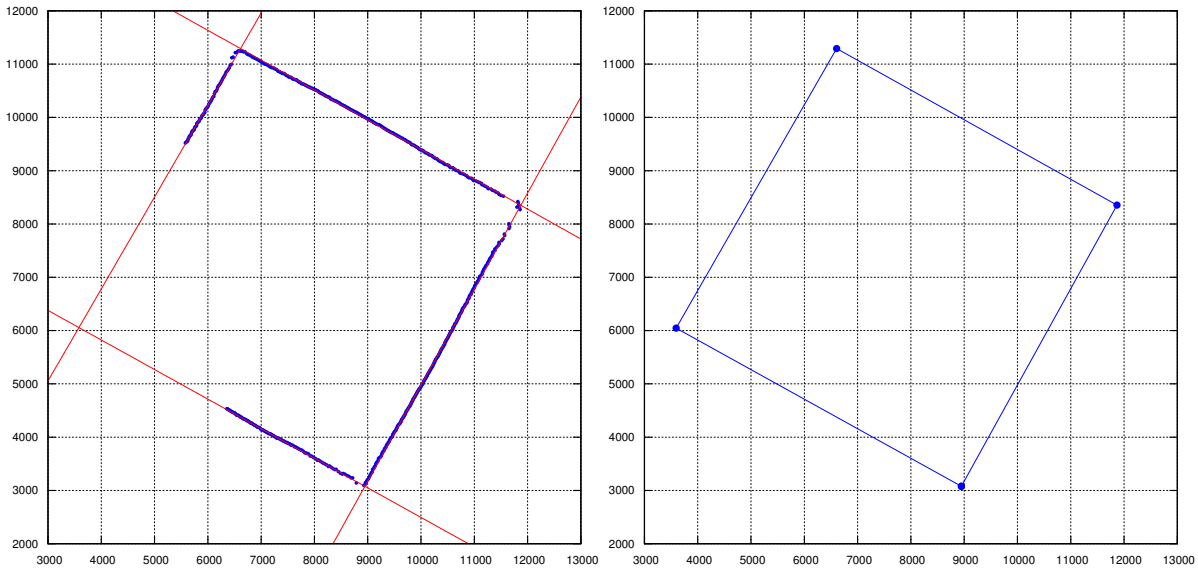
7 Zusammenfassung

In diesem Kapitel werden die Ergebnisse dieser Arbeit zusammengetragen und neue Upgrade-Möglichkeiten für die Weiterentwicklung des mobilen Roboters “Wedico” vorgeschlagen.

7.1 Ergebnisse

Hardware-Tests

Nachdem die wesentlichen Schritte anhand von Simulationsmodellen erläutert und dargestellt wurden, folgen nun einige Bilder aus Hardware-Tests zur Verifikation der simulierten Algorithmenschritte.



(a) Rohscan mit Geradenerkennung

(b) Fertiges Referenzpolygon

Abbildung 7.1: *Kartenerstellung mit Hilfe des Laserscanners Hokuyo URG-04LX-UG01 auf dem FAUST-Testgelände*

Abb. 7.1 zeigt den Algorithmus während der Initialisierungsphase beim Erstellen des Referenzpolygons. Die Scans dafür werden aus einer günstigen Startposition aus durchgeführt, um alle vier Wände erkennen zu können. Diese Startposition gilt ab sofort als interner Koordinatenursprung für den mobilen Roboter.

Nach erfolgreicher Initialisierung der Weltkarte (d.h. dem Referenzpolygon) wird die Position des Laserscanners zu Testzwecken etwas verschoben und verdreht (die mobile Plattform befand sich zur Zeit des Tests noch im Aufbau und stand daher nicht zur Verfügung). Der Folgescan nach der Posen-Änderung ist in Abb. 7.2 zu sehen. Analog zu 6.4 und 6.5 sind hier grün hervorgehoben die Repräsentantenpunkte der einzelnen Punktesegmente dargestellt. Punktesegmente, die nicht als Geraden erkannt werden (d.h. Anzahl Punkte $<$ `MIN_POINTS_ON_LINE`) werden ignoriert. In dieser konkreten Stellung wurden alle Segmente erkannt — der Algorithmus kommt nach der Initialisierung jedoch auch mit weniger (≥ 2) erkannten Geraden aus.

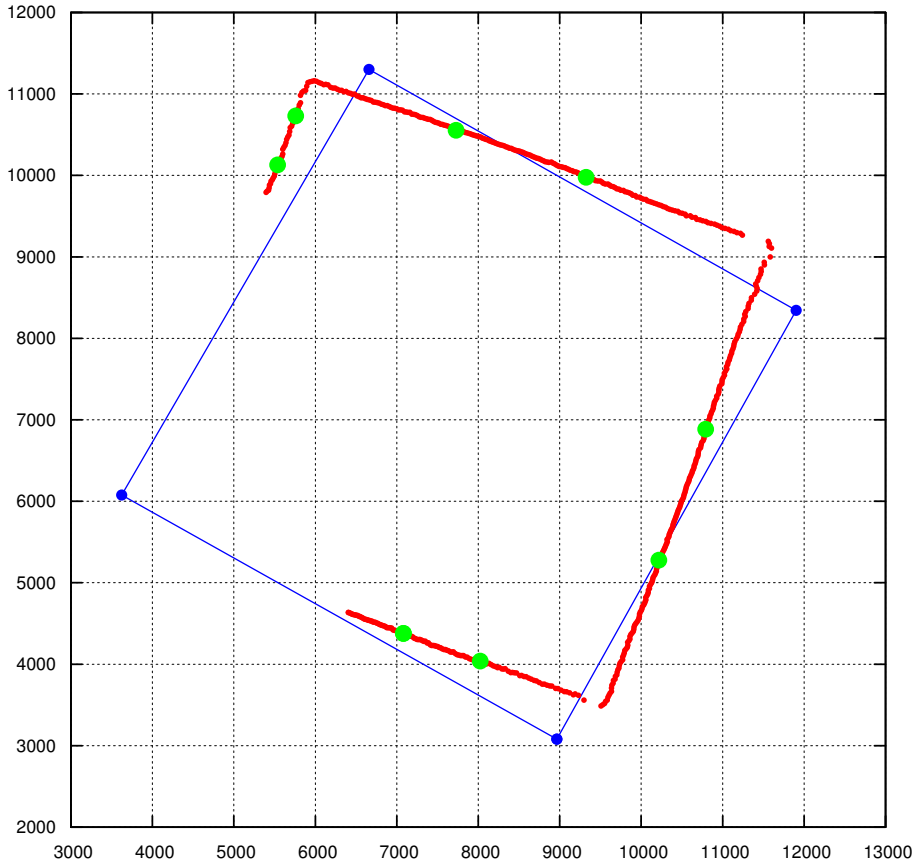


Abbildung 7.2: *Folgescan aus etwas verschoben und verdrehter Position. Grün dargestellt sind die durch Geradenerkennung ermittelten Repräsentantenpunkte, welche an Stelle der rohen Messdaten (rote Punkte) zur Positionsbestimmung verwendet werden.*

Zeitmessungen

Dass das entwickelte Verfahren basierend auf Repräsentanten eine ebenso zuverlässige Positionsbestimmung ermöglicht, wie ein Algorithmus der auf den rohen Messdaten arbeitet, wurde durch ausgiebige Vergleichstests in der Simulation bestätigt. Tabelle 7.1 zeigt drei dieser Testfälle, jeweils einmal mit Berechnung anhand von Rohdaten und einmal mittels Ausgleichsrechnung und Repräsentantenwahl. Zusätzlich wurden noch die Varianten mit vorzeitigem Schleifenabbruch (vgl. Abschnitt 5.3.1) und ohne gegenübergestellt. Der Suchraum wurde wie folgt festgelegt:

Rotation: $[-8° \ldots +8°]$ Auflösung $0.5°$

X-Translation: $[-10mm \ldots +10mm]$ Auflösung $1mm$

Y-Translation: $[-10mm \ldots +10mm]$ Auflösung $1mm$

Transformation	**Rohdaten** o. Schleifenabbruch	**Repräsentanten** o. Schleifenabbruch	**Repräsentanten** m. Schleifenabbruch
$(0, 0, 0°)^T$	$(1, -1, 0°)^T$ $ca. 3.8s$	$(-1, -1, 0°)^T$ $36ms$	$(0, -1, 0°)^T$ $31ms$
$(-8, -8, -5°)^T$	$(-7, -9, -5°)^T$ $ca. 3.8s$	$(-9, -8, -5°)^T$ $34ms$	$(-10, -8, -5°)^T$ $28ms$
$(5, -2, 0°)^T$	$(4, -3, 0°)^T$ $ca. 3.8s$	$(5, -1, 0°)^T$ $34ms$	$(6, -2, 0°)^T$ $30ms$

Tabelle 7.1: *Zeitmessungen (ohne Laserscanner-Messdauer) und ermittelte Transformationen bei einmaligem Algorithmendurchlauf für verschiedene gesuchte Transformationen*

7.2 Fazit

Das Ziel dieser Arbeit war es, für den Neuaufbau des “Wedico”-Trucks im Kontext der FAUST-Projektgruppe der HAW-Hamburg geeignete Positionsbestimmungs-Algorithmen zu analysieren und zu bewerten, sowie einen auf die zeitlichen und räumlichen Bedingungen im Einsatzumfeld optimierten Algorithmus zu entwickeln. Nach der Einschätzung einiger Algorithmen wurde ein hybrider Ansatz gewählt, der vom Ablauf her dem *occupancy grid*-Verfahren ähnelt, als Datenstrukturen jedoch auf Polygone und rohe Messdaten setzt statt auf Gitter. Als weiteres Unterscheidungsmerkmal wird der Grad der Überdeckung von Referenzscan

und Folgenscan nicht durch zellenweises AND ermittelt, sondern durch Betrachtung des mittleren Abstandes aller Folgescan-Punkte zum Referenzpolygon.

Aufgrund der Ähnlichkeit im algorithmischen Ablauf hat auch das ausgewählte Konzept als Hauptproblem die hohe Zahl an Transformationen in Verbindung mit der hohen Anzahl zu bewegender Messpunkte. Dieser Umstand nötigt letztendlich zu einer Wahl zwischen Aktualisierungsrate oder Genauigkeit der Positionsbestimmung.

Im Rahmen dieser Arbeit wurden Möglichkeiten erörtert dieses Problem abzuschwächen und einen Algorithmus zu entwickeln, der sowohl schnell, als auch mit einer guten Auflösung arbeitet. Dieses Ziel konnte durch gezielte Reduktion der Faktorgrößen erreicht werden, auch wenn sich an der Zeitkomplexität des Verfahrens nichts geändert hat.

Ausserdem konnte die interne Kartenerstellung mittels Ausgleichsrechnung und Schnittpunktsberechnung automatisiert werden, was im Vergleich zu manuell vermessenen *a priori*-Polygonen mehr Flexibilität und leichtere Handhabung bietet. Einzige Bedingung bleibt hier eine günstige Ausgangsposition, von der aus möglichst viele Wände vom Laserscanner erfasst werden können.

7.3 Ausblick

Auch wenn das Lokalisierungssystem durch die selbständige Polygonberechnung in der Lage ist, auf einfache Weise auch in anderen (ausreichend polygonalen) Umgebungen eingesetzt zu werden, gibt es in dieser Hinsicht noch durchaus mehr zu erreichen.

Die Nachteile dieses Ansatzes sind z.B. dass keine echte dynamische Kartographierung geboten wird: Einmal eingeschaltet (initialisiert) misst sich der Algorithmus auf den aktuell sichtbaren Raum ein und behält diese Karte als Referenz bis zur Re-Initialisierung durch Neustart. Befindet sich bspw. eine Wand des Raumes zur Initialisierungszeit ausser Reichweite, so existiert diese Wand in der internen Karte nicht — auch nicht wenn sich das mobile System im weiteren Verlauf dieser Wand so weit nähert dass sie vom Laser erfasst wird. Das bedeutet, dass es Messpunkte aus Folgescans geben kann, die zu keinem Polygonabschnitt des Referenzpolygons zugeordnet werden können.

Nun könnte man für solche Gegebenheiten natürlich wieder auf die manuelle Methode umschalten und den Algorithmus mit einer *a priori*-Karte des großen Raumes versorgen. Das würde zwar das Problem der nicht existierenden Wände lösen, jedoch gelten dann wieder die Nachteile der manuellen Kartenerstellung, welche bereits diskutiert wurden.

SLAM

Die Lösung der Problematik führt zu einer erweiterten Klasse von Navigationsgsverfahren: ***S**imultaneous **L**ocalization **A**nd **M**apping*.

Der Begriff SLAM ist nicht als direkter Algorithmus zu verstehen, sondern Vielmehr als Oberbegriff für die Bestrebung, unbekannte Umgebungen auf autonome Weise zu kartographieren, quasi eine Exploration von beliebigen Einsatzorten. Die Karte wird noch während ihrer Erstellung, d.h. noch bevor sie vollständig ist, zur Selbstlokalisierung des mobilen Systems verwendet. Das würde für das gerade beschriebene Problem der ausser Reichweite liegenden Wände bedeuten, dass der Roboter einfach erstmal mit zwei oder drei erfassten Wänden losfährt, seine Karte jedoch während der Fahrt laufend aktualisiert. Sobald also eine neue Wand vom Laser detektiert wird, erscheint diese auch in der Referenzkarte.

Das SLAM-Konzept besteht aus mehreren, auf verschiedene Weise lösbaren Teilaufgaben (vgl. Riisgard und Blas (2005)):

- Extraktion von Landmarken
- Messdaten-Assoziierung
- Zustandsabschätzung
- Zustandsaktualisierung
- Aktualisierung der Landmarken

Konkrete SLAM-Implementierungen existieren für unterschiedlichste Sensor-Typen und Einsatzzwecke. So gibt es auch hier Algorithmen, die für Indoor-Umgebungen entwickelt wurden, als auch welche für den Aussen-Einsatz.

Herzstück von fast allen SLAM-Verfahren ist der *Extended Kalman Filter (EKF)*, eine quasi Standard-Komponente zur nichtlinearen Zustandsabschätzung. Er besteht aus einem Satz von mathematischen Gleichungen, welche auf effiziente Weise die Berechnung des mittleren Fehlers einer Zustandsschätzung für verschiedenste Arten von Prozessen erlauben. Es ist sogar möglich mit Hilfe des EKF Aussagen über den Zustand (in Vergangenheit, Gegenwart und Zukunft) eines Systems zu treffen, ohne über detailliertes Wissen über dessen Eigenschaften zu verfügen (vgl. Welch und Bishop (2003)). Abb. 7.3 zeigt den EKF im SLAM-Prozessablauf.

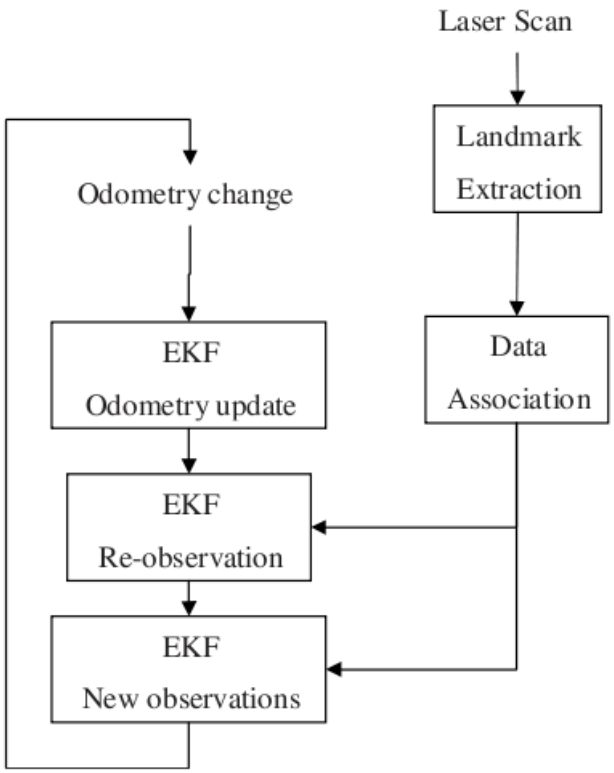


Abbildung 7.3: *Ein SLAM-Prozessablauf (Bildquelle Riisgard und Blas (2005))*

Literaturverzeichnis

[Azenha 2002] AZENHA, Abilio: *Autonomous Vehicles Control and Instrumentation*. 2002. – URL http://paginas.fe.up.pt/~azenha/FCT/WMR1.pdf

[Borenstein 1998] BORENSTEIN, J.: *Experimental Results from Internal Odometry Error Correction with the Omnimate Mobile Robot*. IEEE Transactions: Robotics and Automation. 1998

[Christensen und Fogh 2007] CHRISTENSEN, Rolf ; FOGH, Nikolaj: *Inertial Navigation System*. Aalborg University. 2007. – URL http://www.control.aau.dk/uav/reports/08gr1030a/08gr1030a_student_report.pdf

[Damm 2004] DAMM, Tobias: *Skript zur Vorlesung Praktische Mathematik, Kap. 4*. TU Kaiserslautern. 2004

[Hokuyo] HOKUYO: *URG library*. Hokuyo Automatic Co. Ltd.. – URL http://www.hokuyo-aut.jp/02sensor/07scanner/download/urg_programs_en/lib_tutorial_page.html

[Kind 2003] KIND, Andreas: *Positionsbestimmung*. Seminar Mobile Systeme, Universität Koblenz-Landau. 2003. – URL http://www.uni-koblenz.de/~agrt/lehre/ss2003/seminar/andreas_kind.pdf

[Kraetzschmar u. a. 2004] KRAETZSCHMAR, Gerhard K. ; GASSULL, Guillem P. ; UHL, Klaus: *Probalistic Quadtrees for Variable-Resolution Mapping of Large Environments*. University of Ulm, Neuroinformatics. 2004. – URL http://citeseerx.ist.psu.edu

[Kuipers und Byun 1991] KUIPERS, Benjamin ; BYUN, Yung-Tai: *A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations*. University of Texas, Department of Computer Science. 1991. – URL ftp://ftp.cs.utexas.edu/pub/qsim/papers/Kuipers+Byun-jras-91.pdf

[Lu 1995] LU, Feng: *Shape Registration Using Optimization for Mobile Robot Navigation*. University of Toronto, Department of Computer Science. 1995. – URL web.cs.dal.ca/~eem/cvWeb/pubs/LUFENG_THESIS.pdf

[Lu und Milios 1997] LU, Feng ; MILIOS, Evangelos: *Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans*. York University, Department of Computer Science. 1997. – URL http://citeseerx.ist.psu.edu

[Meisel WS 2010/11] MEISEL, Andreas: *Vorlesung Robot Vision*. Hochschule für Angewandte Wissenschaften Hamburg. WS 2010/11. – URL http://www.informatik.haw-hamburg.de/fileadmin/Homepages/ProfMeisel/Vorlesungen/WP_RobotVision/V/RV06.pdf

[Okubo u. a. 2009] OKUBO, Yoichi ; YE, Cang ; BORENSTEIN, Johann: *Characterization of the Hokuyo URG-04LX Laser Rangefinder for Mobile Robot Obstacle Negotiation*. 2009. – URL www-personal.umich.edu/~johannb/Papers/paper151.pdf

[Pinl 2004] PINL, Markus: *Positionsbestimmung*. Seminar Mobile Systeme, Universität Koblenz-Landau. 2004. – URL www.uni-koblenz.de/~zoebel/ws2004/Positionsbestimmung.pdf

[Rells 1978] RELLS, Karl J.: *Klipp und klar, 100x Luftverkehr*. Bibliographisches Institut Mannheim, 1978. – ISBN 3-411-01712-0

[Riisgard und Blas 2005] RIISGARD, Soeren ; BLAS, Morten R.: *SLAM for Dummies, A Tutorial Approach to Simultaneous Localization and Mapping*. 2005. – URL http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

[SCIP2.0] SCIP2.0: *Communication Protocol Specification For SCIP2.0 Standard*. Hokuyo Automatic Co. Ltd.. – URL http://www.hokuyo-aut.jp/02sensor/07scanner/download/data/URG_SCIP20.pdf

[Tomaszek 2003] TOMASZEK, Wilm: *GPS - Seminar Mobile Systeme*. Seminar Mobile Systeme, Universität Koblenz-Landau. 2003. – URL www.uni-koblenz.de/~agrt/lehre/ss2003/seminar/wilm_tomaszek.pdf

[Tschisow 2008] TSCHISOW, Johann: *Positionsbestimmung für mobile Systeme mit Hilfe eines Laserscanners, Mikrokontrollers und intelligenter Analogsensorik*. Hochschule für Angewandte Wissenschaften Hamburg. 2008. – URL http://opus.haw-hamburg.de/volltexte/2008/564/pdf/Bachelorarbeit_Positionsbestimmung.pdf

[Weiss und v.Puttkammer 1995] WEISS, G. ; V.PUTTKAMMER, E.: *A Map Based on Laserscans Without Geometric Interpretation*. 1995. – URL kluedo.ub.uni-kl.de/volltexte/2000/316/pdf/no_series_85.pdf

[Welch und Bishop 2003] WELCH, Greg ; BISHOP, Gary: *An Introduction to the Kalman Filter*. 2003. – URL www.cs.columbia.edu/~drexel/CandExam/Welch_kalman_intro.pdf

[Wikipedia 2011] WIKIPEDIA: *RANSAC-Algorithmus*. 2011. – URL http://de.wikipedia.org/wiki/RANSAC-Algorithmus

[Wood 1999] WOOD, Charles: *The Instrument Landing System*. Flight Simulator Navigation. 1999. – URL http://www.navfltsm.addr.com/ils.htm

[Woxikon 2011] WOXIKON: *Online Lexikon*. 2011. – URL http://www.woxikon.de

[Zhang WS 2010/11] ZHANG, Jianwei: *Vorlesung Intelligente Roboter, Kap. 5*. Universität Hamburg. WS 2010/11. – URL http://tams-www.informatik.uni-hamburg.de/lehre/2010ws/vorlesung/ir/doc/irKap5.pdf

[Zoebel 2003] ZOEBEL, Dieter: *Übungen zum Seminar Mobile Systeme 2003*. Universität Koblenz-Landau. 2003. – URL http://www.uni-koblenz.de/~agrt/lehre/ss2003/umsmaterial/loesung_blatt1.ps

[Zweigle WS 2010/11] ZWEIGLE, Oliver: *Vorlesung Robotik I*. Universität Stuttgart, Institut für Parallele und Verteilte Systeme. WS 2010/11. – URL http://www.ipvs.uni-stuttgart.de/abteilungen/bv/lehre/lehrveranstaltungen/vorlesungen/SS10/Robotik_II_termine/dateien/robotik_8_lokalisierung.pdf

Anhang

Im Folgenden werden zentrale Datenstrukturen und Funktionen der entwickelten DLL, sowie relevante Auszüge aus der URG library API gelistet und erläutert.

Listing 1 zeigt das Interface der DLL zur Benutzer-Anwendung:

Listing 1: DLL-Interface position.h

```
/**
 * Initialisiere Positionsbestimmung.
 * @param cbfunc: Callback-Funktion fuer Anwendung
 * @return EXIT_SUCCESS oder EXIT_FAILURE
 */
int pos_init(void (*cbfunc)(int xpos, int ypos, double phi));

/**
 * Aktualisiere Odometrie-Daten. Diese Funktion wird nach
 * jeder Aenderung einer oder mehrerer der untenstehenden
 * Parameter aufgerufen, z.B. nach einem Lenkbefehl, oder
 * einem Start/Stopp-Befehl etc.
 * @param rpm: Motordrehzahl in Umdrehungen pro Minute
 * @param gear: Gangstellung
 * @param angle: Lenkwinkel in Grad (positiv bei Linkseinschlag,
 * negativ bei Rechtseinschlag)
 */
void pos_updateODM(int rpm, int gear, double angle);

/**
 * Registriere neue Callback-Funktion
 * @param cbfunc: Callback-Funktion fuer Anwendung
 * @return EXIT_SUCCESS oder EXIT_FAILURE
 */
int pos_resetCBfunc(void (*cbfunc)(int xpos, int ypos, double phi));

/**
 * Starte Positionsbestimmung.
 * Es wird laufend die aktuelle Position bestimmt
 * und nach jedem Durchlauf die Callback-Funktion
 * mit den neuen Positionsparametern aufgerufen.
 * @return EXIT_SUCCESS oder EXIT_FAILURE
```

```
 */
int pos_start();

/**
 * Beende Positionsbestimmung.
 */
void pos_stop();
```

Listings 2 und 3 zeigen die Funktionen der Hardware-Abstraktionsschicht für den Laserscanner und dessen Simulation:

Listing 2: scan.h

```
/**
 * Initialisiere Laserscanner und fordere
 * Messdaten an.
 * @return: EXIT_SUCCESS oder EXIT_FAILURE
 */
int las_init();

/**
 * Funktion zum Auslesen des Laserscanner-Buffers
 * @param data: Zielbuffer fuer Messdaten
 * @return: EXIT_SUCCESS oder EXIT_FAILURE
 **/
int las_scan(long *data);

/**
 * Herunterfahren des Lasers und freigeben der
 * USB-Verbindung
 */
void las_stop();
```

Listing 3: scansim.h

```
/**
 * Initialisiere Bildabmessungen
 * @param _height: Bildhoehe (d.h. simulierte Raumlaenge)
 * @param _width: Bildbreite (d.h. simulierte Raumbreite)
 * @return: EXIT_FAILURE wenn ungueltige Parameter, sonst EXIT_SUCCESS
 */
int ssim_init(int _height, int _width);

/**
 * Scanne von gegebener Position aus
 * @param pose: Pose des Laserscanners
 * @param range: Scanwinkel in Grad
```

```
 * @param data: Zielbuffer fuer simulierte Messdaten
 */
int ssim_scan(struct pose, double range, long *data);
```

In Listing 4 sind alle wichtigen Datenstrukturen und Hilfsfunktionen aufgeführt:

Listing 4: util.h

```
/*
 * Beschreibung eines Vektors mit
 * berechneten Hilfsgroessen.
 */
struct vect {
  int x;     // x-Koordinate
  int y;     // y-Koordinate
  int r;     // Betrag
  double phi; // Winkel (rad)
  int skprod; // Skalarprodukt mit sich selbst
};

/**
 * Ausgleichsgerade in HNF
 */
struct hnf {
  int n;         // Anzahl Punkte
  int ifirst;      // Index erster Messpunkt
  int ilast;       // Index letzter Messpunkt
  struct vect rep1; // Repraesentant 1
  struct vect rep2; // Repraesentant 2
  double phi;      // Winkel-Komponente HNF
  int r;         // Abstand zum Ursprung HNF
  double s2;       // Standardabweichung
};

/**
 * FIFO-Element (einfach verkettete Liste)
 */
typedef struct _fifoitem {
  void *payload;
  struct _fifoitem *next;
} *fifoitem_t;

/**
 * Sehr einfach gehaltener FIFO-Speicher
 */
typedef struct _fifo {
  fifoitem_t head;
  fifoitem_t tail;
```

```
  int size;
  void (*put)(struct _fifo *fifo, void *payload);
  void *(*get)(struct _fifo *fifo);
  void (*merge)(struct _fifo *fifo, struct _fifo *f);
} *fifo_t;

/**
 * Positionsparameter der mobilen Plattform
 */
struct pose {
  int x;      // x-Koordinate im KO-System
  int y;      // y-Koordinate im KO-System
  double phi;   // Fahrzeugwinkel relativ zum KO-Ursprung (degree)
};

/**
 * Datenstruktur fuer Odometrie-Parameter
 */
struct odmdata {
  int speed;    // Geschwindigkeit in mm/ms
  double angle; // Lenkwinkel (links: positiv)
  pthread_mutex_t mutex;
};

/**
 * Custom atan2-funktion
 * @param y: Gegenkathete
 * @param x: Ankathete
 * @return: Winkel in rad
 */
double atan3(double y, double x);

/**
 * Ermittle geschaetzte Position
 * @param old: Letzte bekannte Position
 * @param angle: Lenkwinkel (rad)
 * @param s: Gefahrene Strecke
 * @param new: Geschaetzte Position
 */
void util_estPos(struct pose old, double angle,
                 double s, struct pose *new);

/**
 * Initialisiere Laserscanner Rohdaten, d.h. Messdaten in
 * Polar- und karth. Koordinaten relativ zum KO-Ursprung, sowie
 * Vorberechnung benoetigter Hilfsgroessen.
 * @param src: Feld mit Messdaten in Polarform, jeder Index
 * entspricht einem Winkelschritt des Lasers.
```

```
 * @param dst: Zielfeld fuer intialisierte Messdaten.
 * @param len: Anzahl zu verarbeitender Messdaten.
 * @param phi_min: Messwinkel an src[0] relativ zum KO-Ursprung.
 * @return: Anzahl gueltiger Weltkoordinaten, -1 bei Fehler.
 */
int util_initvect(long *src, struct vect *dst,
                  int len, double phi_min);

/**
 * Kopieren eines Vektor-Feldes
 * @param src: Quelldaten
 * @param dst: Zielbuffer
 * @param len: Anzahl zu kopierender Vektoren
 * @return: EXIT_SUCCESS oder EXIT_FAILURE
 */
int util_copyvect(struct vect *src, struct vect *dst, int len);

/**
 * Verschieben eines Vektor-Feldes in entlang der x-Achse
 * @param src: Quelldaten
 * @param dst: Zielbuffer
 * @param offs: Offset fuer die Verschiebung
 * @param len: Anzahl zu verschiebender Vektoren
 * @return: EXIT_SUCCESS oder EXIT_FAILURE
 */
int util_moveX(struct vect *src, struct vect *dst, int offs, int len);

/**
 * Verschieben eines Vektor-Feldes in entlang der y-Achse
 * @param src: Quelldaten
 * @param dst: Zielbuffer
 * @param offs: Offset fuer die Verschiebung
 * @param len: Anzahl zu verschiebender Vektoren
 * @return: EXIT_SUCCESS oder EXIT_FAILURE
 */
int util_moveY(struct vect *src, struct vect *dst, int offs, int len);

/**
 * Verschieben eines Vektor-Feldes in entlang beider Achsen (x-y)
 * @param src: Quelldaten
 * @param dst: Zielbuffer
 * @param offs_x: Offset fuer die Verschiebung in x-Richtung
 * @param offs_y: Offset fuer die Verschiebung int y-Richtung
 * @param len: Anzahl zu verschiebender Vektoren
 * @return: EXIT_SUCCESS oder EXIT_FAILURE
 */
int util_moveXY(struct vect *src, struct vect *dst,
                int offs_x, int offs_y, int n);
```

```
/**
 * Rotieren eines Vektor-Feldes um geschaetzte Laserscanner-Position
 * @param src: Quelldaten
 * @param dst: Zielbuffer
 * @param rad: Um rad rad drehen
 * @param len: Anzahl zu rotierender Vektoren
 * @param p: Rotationszentrum
 * @return: EXIT_SUCCESS oder EXIT_FAILURE
 */
int util_rotate(struct vect *src, struct vect *dst,
                double rad, int len, struct pose p);

/**
 * Reduktionsfilter zum Verringern der Messdaten-Anzahl durch
 * Schwerpunktbildung benachbarter Messdaten.
 * @param src: Quelldaten
 * @param dst: Zielbuffer
 * @param len: Anzahl Messdaten
 * @param r: Reduktionsradius
 * @return: Anzahl resultierender Punkte
 **/
int util_reduce(struct vect *src, struct vect *dst, int len, int r);

/**
 * Medianfilter zum Eliminieren von Ausreissern.
 * @param src: Quelldaten
 * @param dst: Zielbuffer
 * @param len: Anzahl Messdaten
 * @param wsize: Windowsize
 * @return: void
 **/
void util_median(struct vect *src, struct vect *dst,
                 int len, int wsize);

/**
 * Linienfilter zur Erkennung von Linienstrukturen.
 * @param src: Messdaten
 * @param lines: Feld von Ausgleichsgeraden in HNF
 * @param len: Anzahl Messdaten
 * @return: Anzahl erkannter Linien
 */
int util_lines(struct vect *src, struct hnf *lines, int len);

/**
 * Polygonbildung durch Schnittpunktbildung aus Menge
 * von Geraden in HNF.
 * @param lines: Feld von Geraden zur Bestimmung der Polygon-Eckpunkte
```

```
 * @param polygon: Resultierendes Vektor-Feld (Eckpunkte des Polygons)
 * @param nlines: Anzahl eingehender Geraden
 * @return: Anzahl gefundener Schnittpunkte
 */
int util_polygon(struct hnf *lines, struct vect *polygon, int nlines);
```

Listing 5 zeigt die Definition des Datentyps `urg_t` in der Header-Datei `urg_t.h`. Dieser Typ dient der internen Verwaltung von Laserscanner-Parametern (z.B. Scanfrequenz) und Zustandsinformationen. Er wird von vielen Funktionen der URG-Bibliotheken als Eingabeparameter benötigt.

Listing 5: urg_t (Datei: urg_t.h)

```
typedef struct {

  serial_t serial_;               /* Structure of serial control */
  int errno_;                     /* Store error number */
  urg_parameter_t parameters_;    /* Sensor parameter */

  int skip_lines_;    /* Number of lines to be skipped in one scan */
  int skip_frames_;   /* Number of scans to be skipped */
  int capture_times_; /* Frequency of data acquisition */

  urg_laser_state_t is_laser_on_; /* 0 when laser is turned off */

  long last_timestamp_;           /* Final time stamp */
  int remain_times_;              /* remain times of capture */

  char remain_data_[3];
  int remain_byte_;

} urg_t;
```

Die Bibliotheksfunktionen, welche bei der Implementierung der eigenen Abstraktionsschicht verwendet wurden sind in Listing 6 aufgeführt:

Listing 6: urg_ctrl.h

```
/**
 * Connection
 * @param urg: Structure of URG control
 * @param device: Connection device
 * @param baudrate: Baudrate
 * @return: 0 Normal, <0 Error
 */
extern int urg_connect(urg_t *urg, const char *device, long baudrate);
```

```
/**
 * Disconnection
 * @param urg: Structure of URG control
 */
extern void urg_disconnect(urg_t *urg);

/**
 * Returns the number of maximum data obtained in one scan
 * @param urg: Structure of URG control
 * @return: >=0 number of maximum data obtained in one scan, <0 Error
 */
extern int urg_dataMax(const urg_t *urg);

/**
 * Sets number of times the data to be acquired
 * @param urg: Structure of URG control
 * @param times: Number of scan data
 * @return: 0 Normal, <0 Error
 */
extern int urg_setCaptureTimes(urg_t *urg, int times);

/**
 * Request for distance data.
 * Request for distance data of [first_index, last_index].
 * Return all scan data when specified URG_FIRST, URG_LAST.
 * @param urg: Structure of URG control
 * @param request_type: Received data type
 * @param first_index: Index of the first data stored
 * @param last_index: Index of the last received data stored
 */
extern int urg_requestData(urg_t *urg,
                           urg_request_type request_type,
                           int first_index,
                           int last_index);

/**
 * Receive URG data
 * @param urg: Structure of URG control
 * @param data: Storage location of received data
 * @param data_max: Maximum number of data that can be received
 * @return: 0 > Number of data received, <0 Error
 */
extern int urg_receiveData(urg_t *urg, long data[], int data_max);

/**
 * Directs laser to switch on
 * @param urg: Structure of URG control
 * @return: 0 Normal, <0 Error
```

```
 */
extern int urg_laserOn(urg_t *urg);

/**
 * Directs laser to switch off
 * @param urg: Structure of URG control
 * @return: 0 Normal, <0 Error
 */
extern int urg_laserOff(urg_t *urg);
```

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 9. Mai 2011	
Ort, Datum	Unterschrift