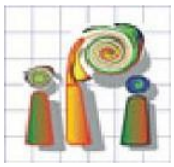

Leibniz Universität Hannover

Fakultät für Bauingenieurwesen und Geodäsie

Institut für Photogrammetrie und GeoInformation

Untersuchungen der Einsatzmöglichkeiten mobiler Roboterplattformen zur Unterstützung von Videoüberwachungssystemen

Masterarbeit
von

B. Sc. Jakob Unger

Hannover 2012

Erstprüfer: Prof. Dr.-Ing. Christian Heipke

Zweitprüfer: Dr.-Ing. Daniel Muhle

Eidesstattliche Versicherung

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit einschließlich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Hannover, den 25. Mai 2012 Jakob Unger

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

Die Betrachtung des Titels dieser Arbeit macht deutlich, dass hier zwei Themen aufeinander treffen. In der Robotik als hochgradig interdisziplinäre Wissenschaft sind viele Elemente aus der Photogrammetrie vertreten, in der Videoüberwachung findet sie ebenfalls Anwendung. Die Geodäsie nimmt als wissenschaftliche Disziplin im Rahmen der Robotik nicht nur im Bereich der Photogrammetrie eine Rolle ein, sondern allgemeiner im Umgang mit Messwerten verschiedenster Sensoren hinsichtlich Fehlerquellen und deren statistischer Auswertung oder auch in Fragen der Kartierung der Umwelt und der Navigation in dieser. In der Videoüberwachung spielen Fragen eine Rolle, die derzeit noch manuelles Eingreifen und den Einsatz von Expertenwissen erfordern. Dazu gehören Herausforderungen wie die Kalibrierung beweglicher Überwachungskameras mit variablem Zoom, die Verknüpfung mehrerer Kameras und ihrer Aufnahmen mit einem Weltkoordinatensystem sowie solche, die sich auf die photogrammetrische Auswertung des Bildinhalts beziehen. Für deren Lösung bietet sich der Einsatz einer mobilen Roboterplattform an.

Yilmaz, Javed & Shah (2006) nennen verschiedene Umstände, die die Objektverfolgung in Videos zu einer komplexen Aufgabe machen. Neben der Verdeckung von Objekten, sind ein möglicher Informationsverlust durch die 2D-Abbildung einer 3D-Szene und Echtzeitanforderung an die Prozessierung, Punkte, die unterstützt werden könnten. Ein Roboter kann sich im Sichtbereich der Kamera bewegen, mittels Sensoren die Umgebung erfassen oder sogar mit ihr interagieren. Nach Verlassen des Sichtbereichs der Kamera, können außerdem über diesen hinaus Daten gesammelt werden. Eine Kartierung der Umwelt durch einen Roboter kann beispielsweise für eine Objekterkennung im Rahmen des Personentrackings hilfreiche Informationen liefern und die Erstellung von Tiefenkarten für Kamerabilder unterstützen. Eine solche Information könnte die Ausrichtung und Lage des Fußbodens relativ zu einer oder mehreren Kameras oder die Vorhersage von Verdeckungen verfolgter Objekte im Kamerabild mithilfe von Verdeckungskarten sein. Liegt eine durch einen Roboter erfasste Karte der sichtbaren Umgebung vor, kann diese beispielsweise Näherungswerte für eine Verdeckungs- oder Tiefenkarte liefern und diese Verfahren somit hinsichtlich der Echtzeitanforderungen beschleunigen. Trägt ein Roboter durch eine geeignete Signalisierung Objektpunkte bei, kann er auch die Kamerakalibrierung unterstützen und helfen, diese zu automatisieren.

Ein Bereich, in dem sich Beispiele für die Anwendung eines Roboters finden, der im Sichtfeld einer Kamera agiert, ist der der Rettungsroboter. Um dieses Thema besteht und wächst ein Forschungsbereich, der sich mit Robotern aller Art beschäftigt, die bei verschiedensten Katastrophen eingesetzt werden sollen (Marsiske, 2011). Ein Beispiel für eine solche Katastrophe ist die von Fukushima: Nachdem ein Erdbeben mit anschließendem Tsunami im März 2011 ein Atomkraftwerk stark beschädigte, traten (und treten) dort atomare Strahlung und verseuchtes Wasser eines solchen Ausmaßes aus, dass seine Umgebung teilweise für Menschen nicht mehr zugänglich ist. Wenn auch erst Wochen nach dem Unglück, kamen hier Roboter zum Einsatz, um Messungen durchzuführen und bebilderte Informationen über den Zustand der Reaktoren zu liefern. Dabei stellte sich die Fernsteuerung der Roboter per Funk als Problem dar, da die Gebäude stark abgeschirmt sind. Es kam ein per Kabel verbundener Unterstützungsroboter zum Einsatz, dessen Kamera ähnlich einer Überwachungskame-

ra einen voraus fahrenden Führungsroboter im Blick hat und die Kommunikation mit diesem ermöglicht (Marsiske, 2012).

Das wachsende Interesse an diesem Bereich lässt sich auch an verschiedenen derzeit ausgeschriebenen Wettbewerben feststellen. So hat beispielsweise jüngst die US-amerikanische Defense Advanced Research Projects Agency (DARPA)[1] einen entsprechenden Wettbewerb ausgeschrieben und bei Roboterwettkämpfen bildet die Rettungsrobotik eine eigene Disziplin[2].

Es sind außerdem Forschungsprojekte in dem Bereich der Rettungsrobotik zu finden, die ein unbemanntes Fluggerät (*unmanned aerial vehicle*, UAV) nutzen, um es über dessen Kamera mit einem am Boden agierenden Roboter zu kombinieren (Marsiske & Kruijff, 2012).

Im Rahmen dieser Arbeit wird ein Projekt geplant, umgesetzt und getestet, das einige der zuvor genannten Aspekte untersucht. Dabei wird ein fahrender Roboter eingesetzt, der mittels verschiedener Sensoren die Umgebung aufnimmt und aus den gesammelten Informationen eine Karte erstellt. Dabei ist die Umgebung zu Beginn der Fahrt unbekannt, so dass keine Planung des zu fahrenden Weges möglich ist. Die Sensoren müssen daher auch für eine Kollisionsvermeidung eingesetzt werden, die parallel zu der Kartierung abläuft. Der Roboter soll von einer beliebigen Position starten können. Die Karte muss aus fehlerbehafteten Beobachtungen zusammengesetzt werden, was über einen wahrscheinlichkeits-basierten Ansatz realisiert wird. Es ist aufgrund des Einsatzes von Low Cost Sensoren mit geringen Genauigkeiten und mehrdeutigen Messungen zu rechnen. Der Roboter bewegt sich im Sichtbereich einer Überwachungskamera, soll anhand einer geeigneten Signalisierung in deren Video verfolgt werden und es soll seine Position im Raum ermittelt werden. Zusätzlich bestimmt der Roboter seine Position selbst, so dass diese redundant vorliegt. Die Kombination der beiden Quellen erfordert eine Zeitsynchronisation.

Für verschiedenste Szenarien erscheint die Nutzung von Überwachungskameras relevant: Sie sind an den verschiedensten Orten montiert und in bestimmten Anlagen sogar vorgeschrieben. Damit stellen sie möglicherweise auch eine bereits vorhandene, spontan nutzbare Infrastruktur für den Einsatz von Robotern dar. Dabei kann eine solche Kamera den Roboter oder der Roboter die Auswertung der Kameravideos, gemäß der einleitend genannten Herausforderungen, unterstützen.

1.2 Aufbau und Kapitelübersicht

Die Arbeit ist in fünf Kapitel gegliedert. Auf diese Einleitung folgt das Grundlagenkapitel. Es enthält neben der Beschreibung der eingesetzten Hard- und Software für die Arbeit relevante mathematische und geometrische Grundlagen und solche aus dem Bereich der Robotik. Das dritte Kapitel schildert die Umsetzung der Arbeit beginnend mit dem Versuchsaufbau bis hin zur Kartierung einer Testumgebung mittels Roboter. Außerdem werden verschiedene Kalibrierungsschritte sowie die konkrete Implementierung des Projekts erläutert. Das vierte Kapitel beschäftigt sich mit den Ergebnissen der in Kapitel 3 beschriebenen Abläufe und Arbeitsschritte. Zusammenfassung und Ausblick runden diese Arbeit in Kapitel 5 ab.

[1] http://www.darpa.mil/NewsEvents/Releases/2012/04/10.aspx, abgerufen am 12.4.2012
[2] http://www.robocup.org/robocup-rescue/, abgerufen am 27.4.2012

2 Grundlagen

Dieses Kapitel erläutert zunächst einige Grundlagen aus den Bereichen Mathematik und Computer Vision, welche zum Verständnis dieser Arbeit nötig sind. Es folgt eine Passage, die sich mit grundlegenden Aspekten der Robotik beschäftigt und weitere Abschnitte über eingesetzte Hard- und Software schließen das Kapitel ab.

2.1 Mathematische und geometrische Grundlagen

2.1.1 Koordinatensysteme

Im Verlauf dieser Arbeit kommen verschiedene kartesische Koordinatensysteme, sowohl zweidimensionale, als auch dreidimensionale, zum Einsatz. Deren Definition und Erläuterung erfolgt in Kapitel 3. An dieser Stelle werden lediglich Grundlagen im Umgang mit diesen erläutert und Bezeichnungen geklärt.

Punkte werden als Vektor geschrieben. Das Kürzel des Koordinatensystems, in dem sie liegen, wird hoch gestellt bezeichnet. Beispielsweise ist ein Punkt $p = [x, \quad y, \quad z]^T$ in einem Koordinatensystem A:

$$p^A \tag{2.1}$$

Die Rotationsmatrix R von einem zweiten ebenfalls dreidimensionalen System B nach A wird folgendermaßen bezeichnet:

$$R_B^A \tag{2.2}$$

Das Zielsystem wird demnach auch hier hoch gestellt geschrieben, das Ausgangssystem tief.

Die Rotation eines Punktes, von dem einen in das andere System wäre also:

$$p^A = R_B^A \cdot p^B \tag{2.3}$$

Die umgekehrte Rotation eines Punktes aus A in das System B entspricht dessen Multiplikation mit der transponierten Rotationsmatrix (2.2).

$$p^B = R_A^B \cdot p^A, \qquad mit\ R_A^B = {R_B^A}^T \tag{2.4}$$

Da die im Folgenden verwendeten Koordinatensysteme auch gegeneinander verschoben und nicht nur verdreht sind, ist eine zusätzliche Translation nötig. Der dreidimensionale Vektor t_B^A, ausgehend vom Ursprung des Ausgangssystems B zum Ursprung des Zielsystems A, repräsentiert die Verschiebungen in den drei Koordinatenrichtungen in B. Wird Formel (2.3) um diese Translation ergänzt, ergibt sich Formel (2.5).

$$p^A = t_B^A + R_B^A \cdot p^B \tag{2.5}$$

2.1.2 Repräsentation von Rotationen

Um Punkte von einem Koordinatensystem in ein anderes zu überführen, gibt es verschiedene Möglichkeiten der Parametrisierung. Um benötigte Rotationen zu beschreiben, sind die im vorausgegangenen Kapitel 2.1.1 bereits genutzten Drehmatrizen mit trigonometrischen Funktionen aus der Photogrammetrie bekannt. Sie beschreiben Rotationen im dreidimensionalen Raum anhand von Dre-

hungen um jede der drei Koordinatenachsen. Da hierbei unter bestimmten Umständen Mehrdeutigkeiten bei der Berechnung der Parameter aus den Elementen der Matrix auftreten, können diese Parameter auch algebraisch (mithilfe von Quaternionen) beschrieben werden. Dies hat außerdem eine schnellere Berechnung zur Folge, weil keine trigonometrischen Funktionen verwendet werden (Luhmann, 2010). Quaternionen sind hier der Vollständigkeit halber genannt, werden jedoch nicht explizit erläutert, da sie in dieser Arbeit nicht zum Einsatz kommen.

Eine andere Möglichkeit ist die Beschreibung einer Rotation um eine im Raum gelagerte Drehachse. Die Achse, um die dabei gedreht wird, sei ein sogenannter Rotationsvektor $r = [r_x \quad r_y \quad r_z]$. Benötigt wird lediglich ein Winkel θ, um den die Rotation um diesen Vektor erfolgen soll. Per Definition entgegen dem Uhrzeigersinn. Da der Rotationsvektor eine Achse vorgibt, ist seine Länge variabel. Die Länge (sein Betrag) kann daher genutzt werden, um diesen Winkel zu repräsentieren:

$$\begin{aligned} \theta &\leftarrow norm(r) \\ r &\leftarrow r/\theta \end{aligned} \tag{2.6}$$

Es werden folglich lediglich drei Elemente benötigt, um eine Drehung im dreidimensionalen Raum zu repräsentieren. Dies entspricht dann auch der Anzahl an benötigten Freiheitsgraden dieser Aufgabe. Die zuvor erläuterte Rotationsmatrix hat hingegen 3x3 Elemente, die die drei nötigen Winkel als Funktion beschreiben (Bradski & Kaehler, 2008).

Um trotzdem mit einer Rotationsmatrix R zu arbeiten, kann diese aus dem Rotationsvektor r mithilfe der Rodrigues-Formel (Hartley & Zisserman, 2003) berechnet werden:

$$R = \cos(\theta) \cdot I + (1 - \cos(\theta)) \cdot rr^T + \sin(\theta) \cdot \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{2.7}$$

Umgekehrt kann aus einer Rotationsmatrix der entsprechende Rotationsvektor berechnet werden, indem (2.8) ausgenutzt wird (Bradski & Kaehler, 2008).

$$\sin(\theta) \cdot \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} = \frac{R - R^T}{2} \tag{2.8}$$

Die Repräsentation als Rotationsvektor wird in der Bibliothek OpenCV (vgl. Kapitel 2.4.1) beispielsweise im Zuge der Kamerakalibrierung und des räumlichen Rückwärtsschnitts verwendet, während in der vorliegenden Arbeit für eigene Berechnungen die Darstellung als Rotationsmatrix vorgezogen wird.

2.1.3 Projektive Geometrie

Eine Alternative zu kartesischen Koordinaten bieten homogene Koordinaten, die in der projektiven Geometrie Anwendung finden. Diese kommt neben der Computergrafik in Photogrammetrie und Computer Vision zum Einsatz. Homogene Koordinaten bieten den Vorteil, Fernelemente wie Punkte und Geraden im Unendlichen repräsentieren zu können. Darüber hinaus lassen sich alle Koordinatentransformationen in Matrizenform formulieren, was die Berechnung anhand von Matrizenmultiplikation erlaubt und damit eine rechnerisch einfache Lösung von Konstruktionsaufgaben ermöglicht. Den Umgang mit ihnen beschreibt Luhmann (2010), der, sofern nicht anders angegeben, die Quelle dieses Unterkapitels ist.

Homogene Koordinaten besitzen eine zusätzliche Dimension, die einem Skalierungsfaktor entspricht. Formel (2.9) zeigt einen vierdimensionalen homogenen Punkt mit seinen Koordinaten x, y und z, dem sogenannten euklidischen Anteil, sowie dem homogenen Anteil w. Er entspricht einem Punkt im dreidimensionalen euklidischen Raum, wenn alle Elemente des Vektors durch w geteilt werden. Dieser Schritt führt dazu, dass $w = 1$ wird, was gleichzeitig dem Skalierungsfaktor euklidischer Koordinaten entspricht. Man spricht von der Normierung der homogenen Koordinate. Das Entfernen der vierten Dimension ergibt den äquivalenten euklidischen Punkt $p = [p_x \quad p_y \quad p_z]^T$. Der euklidisch nicht darstellbare Sonderfall eines Fernpunktes liegt vor, wenn $w = 0$ ist. Umgekehrt erhält man aus der euklidischen die homogene Darstellung eines Punktes durch Hinzufügen des vierten Elements mit dem Wert 1.

$$p_h = \begin{bmatrix} p_{x_h} \\ p_{y_h} \\ p_{z_h} \\ w \end{bmatrix} \rightarrow p = \begin{bmatrix} p_{x_h}/w \\ p_{y_h}/w \\ p_{z_h}/w \\ w/w \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{2.9}$$

Seien r_{ij} die Elemente der Drehmatrix R und t_i die Elemente der Translation t aus Gleichung (2.5), zeigt (2.10) diese Gleichung in homogener Schreibweise.

$$p^A = T_B^A \cdot p^B$$

$$\begin{bmatrix} x^A \\ y^A \\ z^A \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x^B \\ y^B \\ z^B \\ 1 \end{bmatrix} \tag{2.10}$$

Translation und Rotation sind dabei in einer Matrix zusammengefasst, die außerdem als Element den homogenen Skalierungsfaktor enthält. Nach Hartley & Zisserman (2003) entspricht die maximale Anzahl der Freiheitsgrade dieser sogenannten Transformations- oder Abbildungsmatrix T aufgrund dieses Faktors allgemein der Anzahl an Elementen abzüglich des Faktors[3]. Damit lässt sich beispielsweise auch eine räumliche Ähnlichkeitstransformation realisieren, indem die r_{ij} mit dem hinzu kommenden Maßstab multipliziert werden. Die Inversion der Transformation entspricht der Inversen. Werden Koordinaten eines räumlichen Objektes in eine Ebene abgebildet, also im Fall einer projektiven Transformation, ist eine Rücktransformation nicht möglich. T ist in diesem Fall singulär.

2.1.3.1 Einsatz in der Photogrammetrie

Auch die photogrammetrischen Abbildungsgleichungen anhand eines Modells der Zentralprojektion lassen sich in einer Matrix darstellen. Dabei erfolgt die Umrechnung der Objektkoordinaten X in Bildkoordinaten x' mithilfe der Projektionsmatrix P nach (2.11).

$$x' = P \cdot X \tag{2.11}$$

P ist gemäß Gleichung (2.12) aus einer Kalibriermatrix K sowie den Parametern der äußeren Orientierung (Rotationen R, Translationen X_0) zusammengesetzt und hat die Dimension 3x4. Dies ist nachvollziehbar, da ein 4-dimensionaler homogener Vektor auf einen 3-dimensionalen abzubilden ist.

[3] Im Fall der gezeigten Transformation sind es jedoch nur sechs (3 Rotationen in Form einer Rotationsmatrix mit den Elementen r_{ij} sowie 3 Translationen und einer mit Nullen besetzten letzten Zeile mit dem Faktor 1).

$$P = K \cdot R \cdot [I|-X_0] \tag{2.12}$$

Die Kalibriermatrix K, Formel (2.13), enthält nach Luhmann (2010) die Koordinaten des Projektionszentrums x_0', y_0', die Kamerakonstante c und eine Beschreibung von Maßstabsunterschied und Scherung der Bildkoordinatenachsen über zwei Parameter s', m'. Verzeichnungseffekte werden funktional als ortsabhängige Koordinatendifferenzen angebracht, die mit $\Delta x', \Delta y'$ bezeichnet sind.

$$K = \begin{bmatrix} c & cs' & x_o' + \Delta x' \\ 0 & c(1+m') & y_o' + \Delta y' \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

In OpenCV (siehe Kapitel 2.4.1) hingegen werden anstelle einer Kamerakonstante c zwei unterschiedliche Brennweiten f_x, f_y in Richtung der Bildkoordinatenachsen zugelassen (Laganière, 2011). Diese enthalten damit auch den Skalierungsfaktor. Die Scherung s' bleibt unberücksichtigt. Dieser Ansatz entspricht auch dem von Hartley & Zisserman (2003), die die Scherung mit 0 annehmen, da diese bei digitalen Aufnahmesystemen in der Regel nicht auftritt. Die so gebildete Matrix (2.14) wird in OpenCV als Kameramatrix bezeichnet.

$$K_{CV} = \begin{bmatrix} f_1 & 0 & x_o' \\ 0 & f_2 & y_o' \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

Zusammen mit den in einem Vektor angegebenen Koeffizienten der Verzeichnungsfunktionen für beide Koordinatenrichtungen, ist sie das Ergebnis einer Kamerakalibrierung (siehe Kapitel 3.6.1).

2.1.3.2 Homographie

Nach Hartley & Zisserman (2003) ist eine Homographie eine umkehrbare geradentreue Abbildung des zweidimensionalen homogenen Raumes auf sich selbst. Liegen drei Punkte auf einer Linie, tun sie dies nach Anwendung der Abbildung demnach auch noch. Synonym verwendete Begriffe für eine solche Abbildung sind Projektivität und Kollineation. Luhmann (2010) beschreibt das Prinzip ohne homogene Koordinaten unter dem Begriff der ebenen projektiven Transformation.

Eine Homographie wird in der projektiven Geometrie durch eine 3x3-Matrix H beschrieben. Diese beschreibt die Abbildung eines Punktes x in einer Ebene (3x1 homogener Vektor) gemäß Formel (2.15) in eine zweite, die verschoben, gedreht und verzerrt sein kann.

$$x' = Hx$$
$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.15}$$

Wie schon im Fall der Abbildungsmatrix T aus Formel (2.10) entfällt ein Freiheitsgrad der Homographie auf den Skalierungsfaktor. Wird dieser geändert, indem die gesamte Matrix skaliert wird, bleiben deren projektive Abbildungseigenschaften gleich. Um die verbleibenden 8 Freiheitsgrade abzudecken, sind mindestens vier (2D)-Punkte, die sowohl in dem Ziel- als auch in dem Ausgangssystem vorliegen, nötig (Hartley & Zisserman, 2003).

Beispielsweise können mithilfe der Projektion Punkte auf die Bildebene abgebildet werden, die in dem Objektkoordinatensystem in einer Ebene liegen. Eine Homographie stellt somit eine Vereinfachung der 3x4 Projektionsmatrix aus Formel (2.11) für diesen Fall dar. Abbildung 2.1 zeigt dieses Beispiel einer Homographie. Die Ebene im Objektraum könnte beispielsweise eine Fassade sein oder ein Schachbrettmuster (siehe Kapitel 3.3). Für den Fall rechtwinkliger Ebenenkoordinaten x, y sind sechs Freiheitsgrade ausreichend: Drei Translationen, drei Rotationen, keine Scherungen.

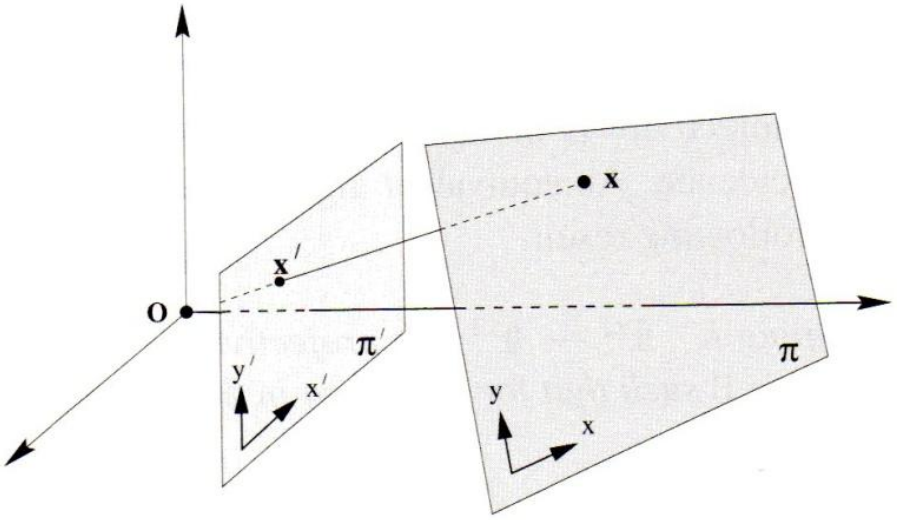

Abbildung 2.1: Abbildung eines Punktes einer Ebene in eine andere im Sinne einer Zentralprojektion
(Bildquelle: Hartley & Zisserman, 2003)
Abbildung des Punktes x in der Ebene π auf die Ebene π'. x, y bzw. x', y' sind die lokalen 2D-Koordinatensysteme der Ebenen. O ist das Projektionszentrum.

2.1.4 Rasterung von Linien

Bei der Rasterung von Linien geht es darum, eine Linie auf ein Punktraster abzubilden, also die Punkte eines Rasters zu markieren, die die Linie am besten wiedergeben. Geometrisch wird dabei anhand des Abstands der jeweils nächsten Rasterpunkte zur Gerade entschieden, welcher markiert wird Dies ist der, der dichter an ihr liegt. Die Vorgehensweise ist also nutzbar, um die zwischen Roboter und detektiertem Hindernis auf einer Gerade liegenden Felder in einer Rasterkarte beispielsweise nach einer Ultraschallmessung als frei zu markieren (siehe Kapitel 3.5).

Neben dem naheliegenden Ansatz von der Geradengleichung ausgehend, die Rasterschritte durchzugehen und die Werte zu runden, gibt es effizientere Lösungen dieser Aufgabe. Effizient bedeutet hier, dass die Berechnung ohne Multiplikation und Division implementiert werden kann. Eine frühe Veröffentlichung stammt von Bresenham (1965), der ein Verfahren schildert, das die Steuerung eines digitalen Plotters zum Zeichnen einer geraden Linie durch Bewegungen in einem festen Raster ermöglicht. Ausgehend vom Anfangspunkt der Gerade wird das umliegende Raster in Oktanten eingeteilt (Abbildung 2.2).

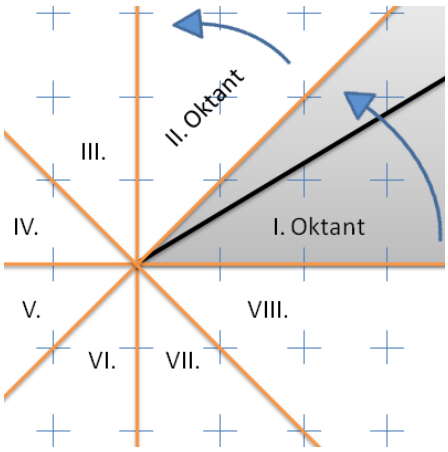

Abbildung 2.2: Die zu rasternde Linie
Linie in schwarz mit den Oktanten I bis VIII

Die Berechnung für eine Gerade wird im ersten Quadrant gezeigt und lässt sich auf die anderen Quadranten übertragen. Da der Algorithmus in der Computergrafik für die Darstellung von Linien im Pixelraster von Bildschirmen verwendet wird, finden sich viele neuere Quellen mit unterschiedlichen Verbesserungen und Varianten des Verfahrens, die darauf aufbauen. Pitteway (1967) löst die Aufgabe mit demselben Ziel auf einem Weg, der neben Geraden auch beliebige Kurvenstücke von Kegelschnitten auf ein Raster überträgt. Der Ablauf gestaltet sich wie folgt:

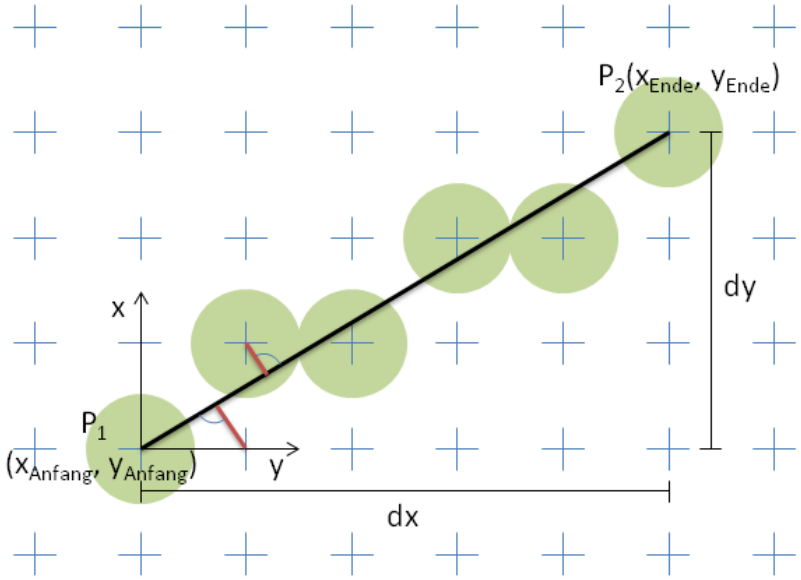

Abbildung 2.3: Gerasterte Linie
Zu rasternde Linie (schwarz) mit der zu minimierenden Distanz der ersten in Frage kommenden Rasterpunkte (rot) zu dieser. Gerasterte Linie in grün.

Zunächst sind die benötigten Rasterschritte (dx und dy) zwischen Anfangspunkt P_1 und P_2 zu bestimmen. Diese ergeben sich aus den Koordinatendifferenzen in einem in P_1 gelagerten, lokalen Koordinatensystem (Abbildung 2.3).

Ein Schritt um eine Ganzzahl in die größere Richtung, also in die, in der mehr Schritte zu P_2 nötig sind (hier x), ist ein sogenannter „schneller“ Schritt, einer in die kleinere Richtung (hier y) ein „langsamer“ Schritt. Es wird nun ausgehend von P_1 immer ein schneller Schritt gemacht. Gleichzeitig wird überprüft, ob ein langsamer Schritt nötig ist. Dies geschieht anhand einer mitgeführten Fehlervariable e. Dieser wird pro schnellem Schritt der Wert der langsamen Richtung (hier also dy) abgezogen. Sinkt ihr Wert unter null, ist ein langsamer Schritt nötig und der Fehlervariable wird der Wert der schnellen Richtung (hier dx) addiert, bevor mit dem nächsten schnellen Schritt fortgefahren werden kann. Der Startwert von e ist der halbe Wert der schnellen Richtung (hier also dx/2). Dies gilt auch, wenn für Start- bzw. Endpunkt Gleitkommawerte zugelassen werden, solange dann alle Berechnungsschritte des Algorithmus unter Berücksichtigung der Nachkommastellen ablaufen.

Im Beispiel (Abbildung 2.3) sinkt die Fehlervariable also bereits nach dem ersten schnellen Schritt in x-Richtung unter null, was bedeutet, dass ein langsamer Schritt in y unternommen und dx auf e addiert wird. Im nächsten Schritt bleibt e dann allerdings auch nach Abzug von dy positiv, weshalb der Rasterpunkt unterhalb der Linie markiert ist.

Neben der Betrachtung von schnellen und langsamen Schritten, werden die Vorzeichen von dx und dy genutzt, um die Berechnung in dem Oktant durchzuführen, in dem die jeweilige Gerade liegt. Die Wahl des Oktanten erfolgt dann durch Fallunterscheidungen (Bresenham, 1965).

2.2 Robotik

Im Folgenden wird auf einige allgemeine, grundlegende Aspekte der Robotik eingegangen und es werden für die vorliegende Arbeit wichtige Themen erläutert.

Die Geschichte und Entstehung der Robotik werden von Siciliano & Khatib (2008) behandelt. Demnach stammt der Begriff des „Roboters“ von dem slawischen Wort „robota“, welches untergeordnete Arbeit bezeichnet. Er wurde eingeführt von dem Bühnenautor Karel Čapek in einem Bühnenstück im Jahr 1920. 1940 verwendete der Autor Isaac Asimov den Begriff in seinen Romanen und beschrieb drei Grundsätze für die Interaktion zwischen Robotern und Menschen. Während noch weiter zurück reichende Beispiele für Roboter genannt werden, begann die reale Umsetzung erster Roboter erst mit dem Fortschritt im Bereich der Informatik ab Mitte der 1950er Jahre (Berns & Schmidt, 2010). Auch heute finden sich in Literatur und Film Roboter, die der realen Entwicklung weit voraus sind und das Verständnis des Begriffs neben den real existierenden Umsetzungen prägen.

Berns & Schmidt (2010) beschreiben den Roboter aus wissenschaftlich-technischer Sicht als Maschine, die beispielsweise über Arme, Räder, Beine, Sensoren, Steuer- und Auswerteelektronik verfügt und damit die Umwelt erfassen und mit ihr interagieren kann. Kapitel 2.3.1 beschreibt die für diese Arbeit eingesetzte Hardware, die typische Elemente eines Roboters umfasst.

2.2.1 Einteilung von Robotersystemen

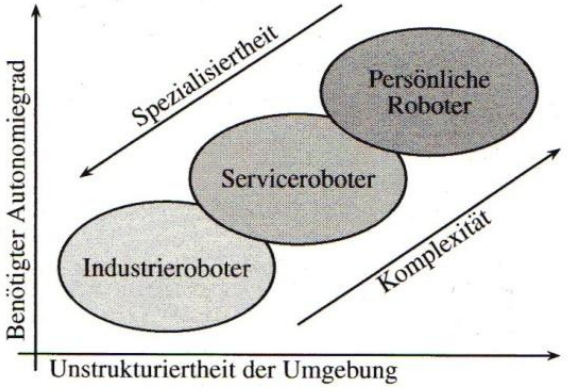

Abbildung 2.4: Klassen von Robotern
(Quelle: (Berns & Schmidt, 2010))
Je unstrukturierter die Umgebung und je höher der benötigte Grad an Automatisierung, desto weniger spezialisiert und desto komplexer wird ein Roboter.

Berns & Schmidt (2010) beschreiben eine Einteilung heutiger Robotersysteme in drei grobe Klassen, die sich auf deren Einsatzgebiet und Anwendung beziehen. Auch Siciliano & Khatib (2008) beschreiben Industrieroboter, also beispielsweise Greifarme, die in der Autoindustrie für die präzise und automatische Fertigung eingesetzt werden oder mit radioaktivem Material umgehen können, als die älteste Klasse von Robotern. Die nächste Klasse bilden Roboter, die über Sensoren die Umgebung wahrnehmen und sich dadurch in dieser bewegen und Dienstleistungen für den Menschen erbringen können. Neben Forschungsrobotern wie Raumsonden oder Robotern zur Kanalinspektion werden auch solche aus dem Unterhaltungs- und Bildungsbereich genannt. Ein aktuelles Beispiel sind Saug- und Wischroboter für den Privathaushalt. Der Bereich der persönlichen Roboter umfasst Menschenähnliche beziehungsweise solche, die weniger spezialisiert sind und allgemein Aufgaben des Alltags in der natürlichen Umgebung des Menschen übernehmen können. Hier finden sich eher Beispiele in

Filmen als in der Realität, da dieser Bereich ein aktuelles Forschungsthema ist (Berns & Schmidt, 2010).

2.2.2 Repräsentation von Karten

Ein Roboter, der seine Umgebung erfassen und sich in ihr bewegen können soll, benötigt eine Karte als Repräsentation dieser Umwelt. Diese liegt oftmals nicht vor oder ist nicht aktuell, so dass die Messwerte verschiedener Sensoren des Roboters für die Kartierung eingesetzt werden. Dabei wird der Roboter als in sich steifer Körper angesehen, dessen sogenannte Pose seine Koordinatenposition und seine Ausrichtung in einem Welt- beziehungsweise Objektkoordinatensystem beschreibt. Auf welchen Punkt des Roboters sich die Position bezieht, ist festzulegen. Häufig wird die Mitte einer Antriebsachse oder das Drehzentrum verwendet. Die Ausrichtung wird meist als Drehung zu der Achse der Geradeausfahrt des Roboters beschrieben. Eine Pose besteht allgemein also aus einer Drehmatrix und einem Translationsvektor. Im zweidimensionalen Fall handelt es sich lediglich um eine Richtung, sowie X- und Y-Koordinate.

Um Karten der Umwelt zu erstellen und zu repräsentieren, gibt es verschiedene Möglichkeiten, die Siciliano & Khatib (2008) und Berns & Schmidt (2010) beschreiben. Letztere unterteilen sie in drei Kategorien. Abbildung 2.5 zeigt diese drei Repräsentationen ein und derselben Umgebung:

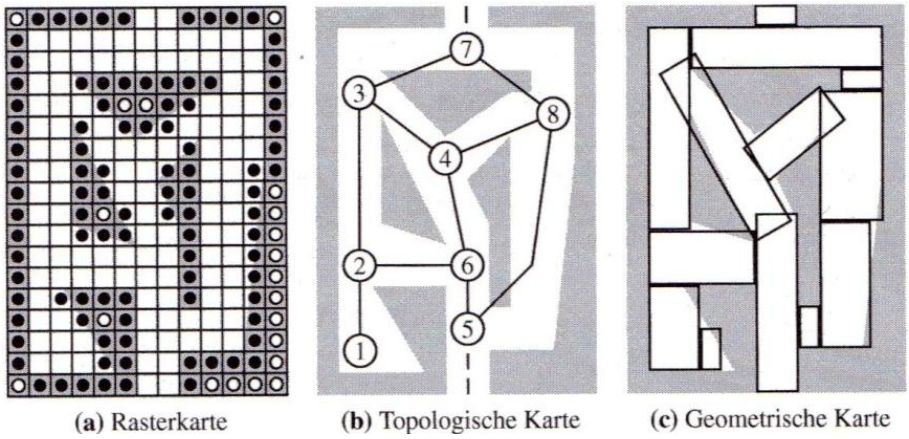

Abbildung 2.5: Kartenrepräsentation
(Quelle: (Berns & Schmidt, 2010))
(a) Einteilung der Umwelt in feste Rasterzellen, die beinhalten, ob die Zelle frei oder belegt ist. (b) Speicherung markanter Merkmale als Graph mit Knoten und Kanten. (c) Zusammenfassung erfasster Merkmale zu geometrischen Elementen

Die **Rasterkarte** (im Englischen unter dem Begriff „*grid map*“ zu finden) basiert auf einer festen Einteilung der Umwelt. Jede Rasterzelle gibt dabei den Zustand dieses Bereiches an, also beispielsweise, ob dieser belegt ist. Nicht erreichbare oder nicht erfasste Bereiche haben dabei ebenfalls einen Wert – in der Abbildung sind sie mit einem weißen Punkt dargestellt. Der Belegtheitszustand kann auch über eine Wahrscheinlichkeit angegeben werden. Beispielsweise bekommen alle Zellen den Startwert 0 und man erhöht den Wert, falls eine Belegung erfasst wird, beziehungsweise verringert ihn, falls eine Belegung durch eine Messung unwahrscheinlicher wird (mehr dazu in Kapitel 3.5). Siciliano & Khatib (2008) beschreiben solche auf Wahrscheinlichkeiten basierenden Raster unter dem Begriff „*Probabilistic Grid*“. Diese werden als die einfachste Umsetzung der Bayes’schen Regel für die Zusammenführung von Sensordaten genannt[4]. Eine Navigation von einer Rasterposition zu einer anderen erfolgt über die Suche des kürzesten Weges über freie Rasterzellen. Je detaillierter die Umwelt zu

[4] Eine weitere Möglichkeit bildet danach das auch aus der Geodäsie bekannte Kalman-Filter.

erfassen ist, desto höher muss das Raster aufgelöst werden und desto mehr Speicherplatz wird benötigt. Die Speicherung weiterer Informationen, wie Temperatur und Art des Untergrundes, pro Zelle ist außerdem möglich (Siciliano & Khatib, 2008).

Die **topologische Karte** spart Speicherplatz, indem Merkmale der Umwelt mittels eines Graphen symbolisch abstrahiert werden. Beispielsweise sind Kreuzungen und Sackgassen Knoten und mögliche Verbindungen die Kanten. Gewichte der Kanten, anhand deren Länge oder der möglichen Geschwindigkeit bei der Befahrung, ermöglichen hierbei die Ermittlung geeigneter Wege im Sinne einer Navigation. Auch die Möglichkeiten der Graphenrepräsentation gehen über Informationen möglicher Fahrwege also Belegtheitszustände hinaus.

Werden Objekte oder, wie in der Abbildung, freie Bereiche zu Geometrien zusammengefasst, wird von einer **geometrischen Karte** gesprochen. Diese Diskretisierung der Umgebung kann ebenfalls Speicherplatz sparen und ermöglicht eine Filterung und Generalisierung der Umwelt hinsichtlich relevanter Merkmale und Objekte (Siciliano & Khatib, 2008).

2.2.3 Synchronisation

Um eine der in Kapitel 2.2.2 erläuterten Kartenarten zu erstellen, müssen die Messwerte verschiedener Sensoren des Roboters an dessen Pose zum Zeitpunkt der Messung erfasst werden. Bewegt sich der Roboter während der Messungen, muss also für jede eine entsprechende Pose vorliegen, was eine zeitliche Synchronisation erfordert. Im Fall der vorliegenden Arbeit kommt eine Kamera hinzu, deren Videosequenz in Echtzeit (10-15 Bilder pro Sekunde) die Umwelt des Roboters und diesen selbst aufnimmt und dies mit den Messungen des Roboters kombiniert. Auch die Messungen einer solchen Kamera in Form der einzelnen Videobilder unterliegen der Notwendigkeit einer zeitlichen Synchronisation. Ziel ist es, nach Möglichkeit für jede Messung die Zeit in Form eines Zeitstempels zur Verfügung zu haben. Im Folgenden werden verschiedene Möglichkeiten aufgezeigt, die im Rahmen der Arbeit in Betracht gezogen wurden.

Dem Geodäten ist diese Aufgabe beispielsweise aus dem Bereich der Multi-Sensor-Systeme bekannt (Neuner, 2009), die viele Elemente und Aspekte beinhalten, die sich auch in der Robotik wieder finden. Hier werden neben dem PPS-Signal (Pulse Per Second) des Satellitensystems GPS (Global Positioning System) die Zeitübertragung per Funk (Beispielsweise DCF77[5]) und über das Internet per Network Time Protocol (NTP) als Möglichkeiten genannt. Wittke (2011) beschäftigt sich im Zusammenhang mit aktiven Kameranetzen ebenfalls mit der zeitlichen Synchronisation. Dabei werden die Videobilder mehrerer Kameras über gemeinsam aufgenommene visuelle Ereignisse synchronisiert. Vorteil dieser Vorgehensweise ist, dass keine Nachrichten ausgetauscht werden müssen. Das führt bei anderen Verfahren zu Unsicherheiten, da die Übertragungsdauer der Nachrichten bei diesen unsicher ist. Visuelle Ereignisse sind beispielsweise optische Signale durch Lampen oder bewegte Objekte im aufgenommenen Video. Dabei wird ausschließlich die mit Lichtgeschwindigkeit übertragene Bildinformation ausgewertet, so dass ein solches Verfahren ohne verzögerte Zeitübertragung auskommt. Die Zeit, die zwischen Bildaufnahme und Abspeichern eines Bildes inklusive Zeitstempel vergeht, lässt sich laut Wittke (2011) anders als die nicht-deterministischen Verzögerungen bei der Nachrichtenübertragung anhand der Bildrate berücksichtigen.

Die drei wesentlichen Wege, Sensoren unterschiedlicher Messgeschwindigkeit und –intervalle zu kombinieren, sind laut Neuner (2009) der taktgesteuerte, bei dem alle Sensoren einem gemeinsamen

[5] http://www.dcf77.de/, abgerufen am 27.4.2012

Takt folgen, der ereignisgesteuerte und der ereignisorientierte. Während bei dem ereignisgesteuerten Vorgehen ein Ereignis eines Sensors die Messung aller anderen auslöst, erlaubt das ereignisorientierte die unabhängige Aufzeichnung der Messwerte der einzelnen Sensoren in der jeweiligen Geschwindigkeit und mit unterschiedlichem Intervall. Letzteres Vorgehen ist damit das flexibelste macht aber eine Interpolation der Messwerte im Zuge der Zusammenführung der Daten nötig.

Neuner (2009) zeigt auf, dass der Begriff Echtzeit kein System darstellen kann und muss, das ohne Zeitverluste operiert. Vielmehr gilt es, die Verzögerungen abzuschätzen und zu berücksichtigen und Ausfälle einzelner Sensoren in bestimmten Intervallen durch Redundanz und Interpolation abzufangen.

Der für das umgesetzte Projekt gewählte Weg für die Synchronisation wird in Abschnitt 3.5.1 erläutert.

2.2.4 Odometrie

Der Begriff *Odometrie* setzt sich aus den griechischen Wörtern *hodos* für *fahren* und *metron* für *messen* zusammen. Er beschreibt die schon lange praktizierte Methode, die zurückgelegte Strecke und Orientierungsänderungen, beispielsweise eines Fahrzeugs, anhand dessen sogenannter Aktuatoren (Räder, Ketten oder auch Schritte bei laufenden Robotern, etc., siehe Kapitel 2.3.1.2) zu ermitteln. Das Grundprinzip ist dabei die Ableitung der Bewegung des Fahrzeugs durch ein mathematisches Modell der damit einhergehenden Bewegung eines oder mehrerer Aktuatoren. Die Aufsummierung der Bewegungen über die Zeit liefert dann die Position des Fahrzeugs als Funktion der Zeit (Siciliano & Khatib, 2008). Beispielsweise wird im Fall von Rädern deren zurückgelegte Strecke pro Umdrehung in Abhängigkeit des Durchmessers, bei Schritten die bekannte Schrittweite genutzt. Ausgehend von einer bekannten Startpose ist durch diese Vorgehensweise die Position über die Zeit bekannt. Das Vorgehen wird auch als *dead reckoning* bezeichnet.

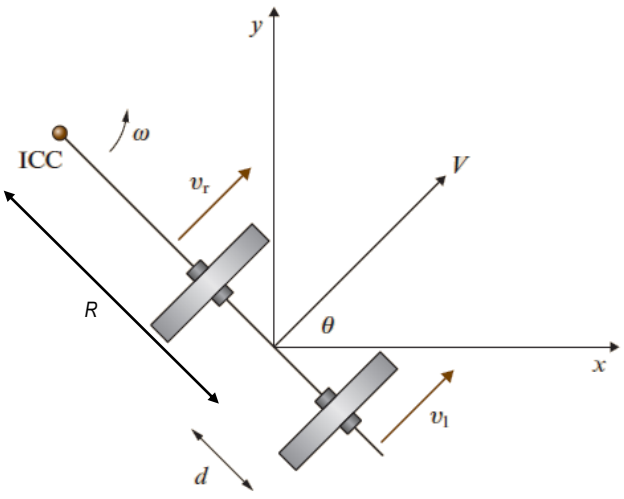

Abbildung 2.6: Differentialantrieb
(Quelle: (Siciliano & Khatib, 2008))
Die Abbildung zeigt linkes und rechtes Rad mit Abstand d zum Mittelpunkt der gemeinsamen Achse und jeweiliger Geschwindigkeit v_r und v_l sowie deren Drehpunkt ICC (instantaneous center of curvature) und Drehrate ω. Die Position des Mittelpunktes zwischen den beiden Rädern wird in einem zweidimensionalen kartesischen Koordinatensystem beschrieben. Die resultierende Geschwindigkeit ist V.

Siciliano & Khatib (2008) beschreiben den Differentialantrieb, der auch im Rahmen dieser Arbeit zum Einsatz kommt, als wohl einfachste Umsetzung eines Antriebs, der die Ermittlung der Pose per Odo-

metrie ermöglicht. Abbildung 2.6 zeigt dessen Prinzip in der Aufsicht. Die beiden Räder können unterschiedliche Umdrehungsgeschwindigkeiten haben, was eine Kurvenfahrt ermöglicht. Der Mittelpunkt zwischen ihnen sei hier das Zentrum des Fahrzeugs. Neben dem Durchmesser der Räder, von dem die zurückgelegte Strecke pro Umdrehung abhängt, wird für die Berechnung auch deren Abstand benötigt (er entspricht $2 \cdot d$ in der Abbildung). ω ist die Rotationsgeschwindigkeit um ein Drehzentrum ICC (für englisch: *instantaneous center of curvature*), dessen Abstand vom Mittelpunkt des Roboters R von der gefahrenen Kurve abhängt: Je unterschiedlicher v_l und v_r sind, desto enger wird die Kurve und desto geringer der Abstand. Im Fall der Geradeausfahrt ($v_l = v_r$) wird dieser Abstand unendlich. Je nachdem welches Rad schneller ist ändert sich die Seite, auf der er liegt (gezeigt ist $v_l > v_r$). Die Geschwindigkeit V ergibt sich als $V = \omega R$. Die Orientierung des Fahrzeugs in dem Koordinatensystem ist durch θ definiert. Ausgehend von einer bekannten Startpose (x_0, y_0, θ_0) ergibt sich die aktuelle Pose durch das Aufaddieren der gemessenen Bewegung auf die vorhergehende Epoche $i - 1$ gemäß Formel (2.16).

$$\begin{aligned} \theta_i &= \theta_{i-1} + \Delta\theta \\ x_i &= x_{i-1} + \Delta x \\ y_i &= y_{i-1} + \Delta y \end{aligned} \qquad \textbf{(2.16)}$$

Eine Herleitung der benötigten Formeln für $\Delta\theta$, Δx und Δy im Fall des Differentialantriebs geben Berns & Schmidt (2010). Neben der bekannten Startposition sind die beiden Geschwindigkeiten der Räder die einzigen benötigten Größen, um anhand des in Formeln modellierten Differentialantriebs zu einem beliebigen Zeitpunkt t eine Pose zu erhalten. Allerdings geht dieses Modell von einer perfekten Umwelt aus. Mögliche Fehlerquellen sind jedoch

- ungenaue Eingabegrößen (Durchmesser der Räder und deren Abstand),
- unsichere Ermittlung der Radumdrehungen (Fehler von Drehgebern),
- ungenaue Ansteuerung der Motoren (befohlene Rotation entspricht nicht der tatsächlichen),
- unzureichende Modellierung des Roboters (beweglicher Körper, Spiel der Räder zu diesem und zueinander)

  …und seiner Umwelt (verformbarer Untergrund, Rutschen der Räder bei Beschleunigungen oder bei der Kurvenfahrt, Breite der Räder nicht berücksichtigt) (Siciliano & Khatib, 2008).

Abbildung 2.7 zeigt eine per Odometrie erhaltene Trajektorie mit den ersten sieben Epochen, die solche Fehler in Form von Fehlerellipsen (gestrichelt) schematisch berücksichtigt. Auch wenn sich einige der genannten Fehler beispielsweise durch Kalibrierung minimieren lassen, werden die Restfehler mit jeder Epoche größer, da die Fehler aller vorausgegangenen Epochen aufaddiert werden. Dieses für die Odometrie typische Fehlerverhalten muss bei deren Einsatz für die Positionsbestimmung berücksichtigt werden und lässt sich nur durch die Hinzunahme anderer Messungen korrigieren (Siciliano & Khatib, 2008). Das Problem ist in der Geodäsie beispielsweise von der Inertialnavigation bekannt, die deshalb häufig mit Satellitenortung kombiniert wird. Die Inertialnavigation bietet, wie die Odometrie, eine relative Positionierung, die Satellitenortung ist analog zu der Messung per Kamera ein absolutes Positionierungsverfahren.

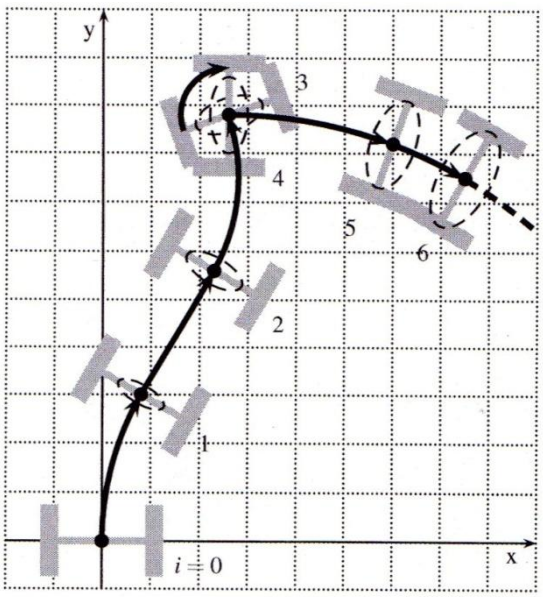

Abbildung 2.7: Odometrie-Fehlerfortpflanzung
Quelle: (Berns & Schmidt, 2010)
Beispielhafte Trajektorie in einem kartesischen, zweidimensionalen Koordinatensystem. Ausgehend von einem Startpunkt (Startepoche *i=0*), der hier im Ursprung liegt, sind die aktualisierten Posen bis *i=6* mit den größer werdenden Fehlerellipsen gezeigt.

2.3 Hardware

2.3.1 LEGO Mindstorms NXT 2.0

Mindstorms NXT 2.0 ist ein Roboterbaukasten der Firma LEGO GmbH, der neben 619 Teilen des Stecksystems Lego Technik einen programmierbaren Mikrocontroller, den NXT-Baustein, und verschiedene Sensoren und Aktuatoren enthält (LEGO Group, 2006). Diese sind drei Servomotoren mit integriertem Rotationssensor, ein Ultraschall-, ein Farbsensor und zwei Berührungssensoren (Abbildung 2.8). Die Bauteile lassen sich mit beliebigen weiteren Lego Technik-Teilen erweitern sowie mit Aktuatoren und Sensoren weiterer Sets kombinieren. Auch ist die Kommunikation zwischen NXT-Mikrocontrollern und mit PC oder Handy möglich. Darüber hinaus bieten die LEGO GmbH und andere Firmen[6] weitere kompatible Sensoren an.

[6] Beispielsweise: http://www.hitechnic.com/, abgerufen am 1.3.2012

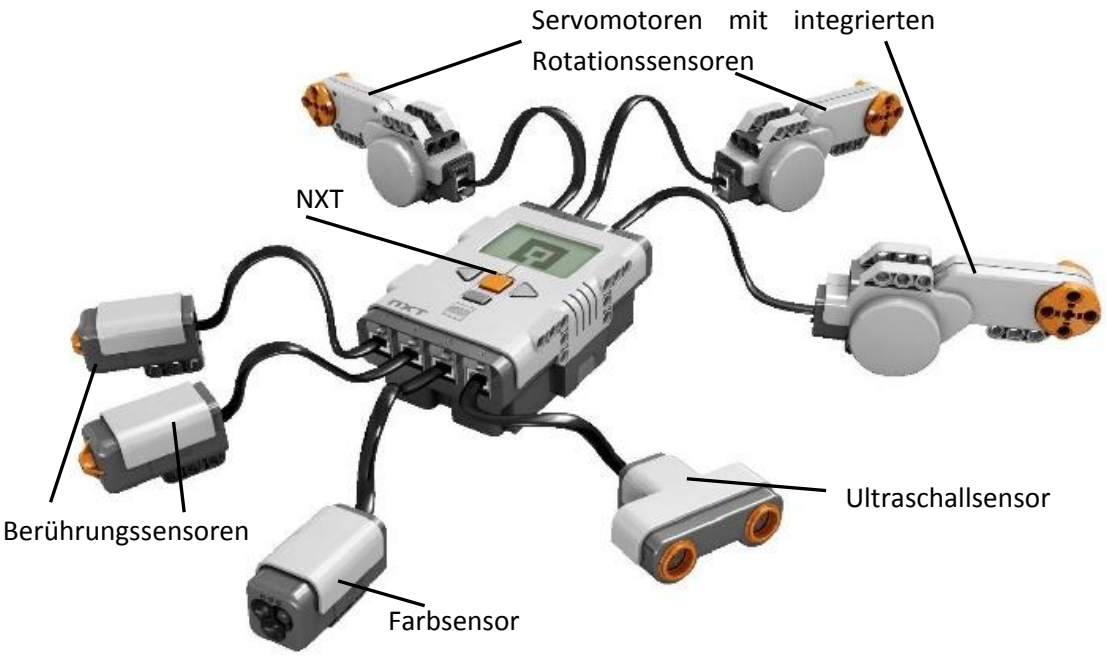

Abbildung 2.8: LEGO Mindstorms NXT 2.0
Das Bild zeigt neben dem NXT-Baustein, die angeschlossenen Motoren und Sensoren des Bausatzes. (Bildquelle: (LEGO Group, 2006))

2.3.1.1 NXT

Der NXT-Baustein (Abbildung 2.9) stellt den Kern des Mindstorms-Systems dar. Seine technischen Daten zeigt Tabelle 2.1. Weitere Details und Angaben zu den Belegungen der Ports, der Funktionsweise der Motoren und der einzelnen Sensoren sowie den Verfahren zur Datenübertragung und Kommunikation beschreiben Berns & Schmidt (2010). Auf die in der vorliegenden Arbeit verwendeten Elemente gehen die beiden folgenden Unterkapitel ein.

Abbildung 2.9: Der NXT-Baustein
mit den in Tabelle 2.1 genannten Elementen
(Bildquelle: http://www.nxt-in-der-schule.de/bilder/nxt.jpg)

Tabelle 2.1: Technische Daten des NXT

Prozessor	32-Bit-Mikrocontroller ARM7, 48 MHz, 256 KB Flash-Speicher, 64 KB RAM
Coprozessor	8-Bit-Mikrocontroller AVR, 4 MHz, 4 KB Flash-Speicher, 512 Byte RAM
Kommunikation	Bluetooth Class II V2.0, USB Full Speed Port mit 12 Mbit/s, RS-485, I2C
Eingangsports	4x, 6-adriges Kabel, analoge und digitale Übertragung (davon 1x RS-485 Port)
Ausgangsports	3x, 6-adriges Kabel, Motorausgänge mit Rückkanal
Bildschirm	LCD, 100x64 Pixeln Auflösung (beispielsweise 16 Zeichen in 8 Zeilen darstellbar)
Bedienelemente	4 Tasten: Ein/Enter, Rechts, Links und Löschen/Zurück/Aus
Lautsprecher	8 kHz mit 8-Bit-Auflösung
Stromquelle	6x AA-Zellenbatterien/-akkus oder Zubehör-Akku

Quellen: (LEGO Group, 2006), (Berns & Schmidt, 2010), (Moral, 2009)

Auf die Kommunikation über RS-485 oder I2C wird hier nicht weiter eingegangen, da sie der Anbindung weiterer Sensoren bzw. der Kommunikation zwischen mehreren NXTs dienen (Moral, 2009), was hier nicht erforderlich ist. USB steht für Universal Serial Bus und bezeichnet die weit verbreitete kabelgebundene Schnittstelle zur Anbindung verschiedenster Geräte an einen Computer. Bluetooth ermöglicht dies per Funk im Bereich von 2,4 GHz über kurze Distanzen (je nach Bluetooth-Klasse bis zu 100 Meter, im Fall des NXT maximal 10 Meter) und stellt einen insbesondere im Mobilfunkbereich weit verbreiteten Standard dar. Detaillierte Angaben zu den Kommunikationsmöglichkeiten des NXT und deren Nutzung über LeJOS (siehe Kapitel 2.4.2) mit einem Schwerpunkt auf der Bluetooth-Schnittstelle liefert Moral (2009).

2.3.1.2 Aktuatoren

Als Aktuatorik werden in der Robotik Elemente bezeichnet, die eine Bewegung hervorrufen oder die Umwelt verformen können. Dies sind beispielsweise Motoren, die eine mechanische Bewegung ermöglichen, Pumpen, die mit Druck arbeiten, oder auch Lautsprecher, die die Umgebung beschallen (Berns & Schmidt, 2010).

Elektromotoren, die elektronische Signale in mechanische Energie umwandeln, sind wichtige Aktuatoren für Robotersysteme der hier verwendeten Größe. Die Lego-Servomotoren enthalten neben dem eigentlichen Elektromotor auch ein Getriebe und einen Rotationssensor. Die Funktionsweise erläutern Berns & Schmidt (2010). Die Ansteuerung erfolgt mittels Pulsweitenmodulation, was bedeutet, dass die Länge von Strompulsen einer festgelegten Frequenz das Drehmoment des Motors und somit dessen Geschwindigkeit vorgibt. Unter Stromimpuls ist ein Wechsel zwischen Versorgungsspannung und Masse zu verstehen. Aus diesem Grund ändert sich die maximale Leistung (und damit die Maximalgeschwindigkeit) des Motors mit schwächer werdenden Batterien des NXTs.

Die Regelung des Motors erfolgt durch Angabe der Motorumdrehungszahl und der dabei gewünschten Leistung. Unter LeJOS (siehe Kapitel 2.4.2) ist alternativ die Angabe der Drehgeschwindigkeit möglich. Sobald der Motor keinen Befehl mehr erhält, wird auch keine Kraft mehr ausgeübt, die ihn beispielsweise gegen wirkende Gravitation in Position hält. Er kann jedoch angewiesen werden, die Position unter Einsatz von Gegenkraft zu halten.

2.3.1.3 Sensoren

Die Einteilung von Sensoren erfolgt häufig nach aktiven und passiven. Während letztere von außen wirkende Messgrößen erfassen, erzeugen aktive Sensoren ein Signal und messen das, was von die-

sem zurück kommt, um auf eine gesuchte Messgröße zu schließen. Siciliano & Khatib (2008) beschreiben eine für die Robotik wichtige Einteilung nach sogenannten propriozeptiven Sensoren, die intern Größen des Roboters selbst erfassen, und solchen, die dessen Umgebung aufnehmen, sogenannte exterozeptive.

Zu der Klasse der exterozeptiven gehört der im Lego-Motor integrierte Rotationssensor. Er ermöglicht laut Berns & Schmidt (2010), die Umdrehungen des Motors mit einer Genauigkeit von einem Grad zu steuern. Dies geschieht über eine an der Motorachse befestigte Encoderscheibe mit zwei wechselnden Hell-Dunkel-Mustern, deren Umdrehung mit zwei Lichtsensoren abgetastet wird. Geschwindigkeit und Abfolge der Hell-Dunkel-Wechsel ergeben Drehrichtung und –geschwindigkeit der Motorachse[7].

Die anderen Sensoren des Bausatzes werden als exterozeptive verwendet beziehungsweise wird auf den Farbsensor an dieser Stelle nicht weiter eingegangen, da er für die vorliegende Arbeit nicht eingesetzt wird.

Der Tastsensor ist ein Beispiel für einen passiven Sensor. In Form von Schaltern an technischen Geräten wie Fernseher, Fernbedienung, Telefon oder im Haus als Lichtschalter spielt diese Klasse von Sensoren eine wichtige Rolle. Die Funktionsweise ist dabei entweder die Herstellung oder die Unterbrechung eines Stromkreises als Reaktion auf das Drücken des Tasters. Nach Berns & Schmidt (2010) ist der Lego-Tastsensor ein schließender: Der Ausgangsstrom beträgt in gedrücktem Zustand 0,0 mA, sonst 2,2 mA. Taster kommen in der Robotik vornehmlich in Form sogenannter Bumper zum Einsatz, die einer Stoßstange entsprechen, die Kollisionen melden kann.

Während mit einem Taster Kollisionen lediglich erkannt werden können, lassen sie sich mit einem Ultraschallsensor verhindern. Als aktiver Sensor tastet er die Umgebung berührungslos ab. Das Messprinzip ist die Laufzeitmessung wie beispielsweise auch bei vielen Laserscannern. Es wird ein Signal ausgesendet und die Zeit gemessen, bis es wieder empfangen wird. Dazu muss es also von Hindernissen zum Empfänger reflektiert werden. Bei dem ausgesendeten Signal handelt es sich um ein für den Menschen unhörbar hochfrequentes akustisches. Für den Fall, dass Sender und Empfänger dieselbe Distanz zum Objekt haben, erfolgt die Berechnung der Strecke s zwischen Sensor und Hindernis nach Formel (2.17). Sie entspricht der halben Laufzeit des Signals Δt multipliziert mit der Schallgeschwindigkeit in der Luft v_{Luft}.

$$s = \frac{1}{2} \cdot \Delta t \cdot v_{Luft} \tag{2.17}$$

Wie Siciliano & Khatib (2008) erläutern, variiert die Schallgeschwindigkeit je nach Temperatur, Druck und Feuchte. Dabei ist Formel (2.18) für die meisten Umgebungen auf 1% genau. Sie nähert empirisch die Schallgeschwindigkeit in Abhängigkeit der Temperatur in Celsius T_C an.

$$v_T = 20{,}05\sqrt{T_C + 273{,}16\frac{m}{s}} \tag{2.18}$$

Setzt man in (2.18) $T_C = 0°$ ein, ergibt sich eine Geschwindigkeit von 331,38 $\frac{m}{s}$. Bei 20° beträgt sie danach 343,29 $\frac{m}{s}$ und weicht damit deutlich ab. Der Einfluss von Druck und Feuchte ist vielfach geringer und wird im Folgenden daher nicht weiter betrachtet. Der der Temperatur lässt sich anhand Formel (2.17) nach Umstellen abschätzen: Der genannte temperaturbedingte Unterschied führt auf einer Strecke von einem Meter zu einer Änderung der Laufzeit um rund 0,0002 Sekunden. Geht man

[7] Das Prinzip entspricht beispielsweise dem Teilkreisabgriff bei elektronischen Tachymetern.

von der Schallgeschwindigkeit bei 0° aus, obwohl bei 20° gemessen wird, weicht die gemessene Strecke dadurch um rund 3,5 cm ab.

Neben einem kontinuierlichen bietet der verwendete Ultraschallsensor einen als *Ping* bezeichneten Messmodus. Während in dem kontinuierlichen Modus eine Messung nach der anderen durchgeführt wird, die im Sinne einer First-Pulse-Messung eine Distanz liefern, wird in diesem Modus ein Signal ausgesendet und bis zu acht zurückkommende Pulse dieses Signals abgewartet. Die zurückgegebenen Distanzen dieses einen Pulses entsprechen dann mehreren unterschiedlich weit entfernten Zielen.

2.3.2 Kamera

Im Rahmen dieser Arbeit wird die Netzwerk-Kamera AXIS 214 PTZ der Firma Axis genutzt. PTZ steht für die englischen Begriffe *pan*, *tilt* und *zoom*, deutet also darauf hin, dass sich diese Kamera schwenken (± 170º Schwenkbereich), neigen (-30° – 90° Neigungsbereich) und zoomen (18-fach optisch) lässt. Sie liefert den Videostrom wahlweise im MPEG-4 Format (Moving Picture Experts Group) oder als Motion JPEG (Joint Photographic Experts Group, kurz MJPEG) in Auflösungen bis 704x576 Pixeln bei 30 oder 25 Bildern pro Sekunde. Der Bildsensor ist ein ¼'' CCD. Ihre Anbindung an einen Computer erfolgt per kabelgebundenem IP-Netzwerk (LAN). Die Einstellung dieser und weiterer Parameter erfolgt mittels Browser per Netzwerkzugriff auf eine Konfigurationsoberfläche (Axis Communications, 2012).

Das umgesetzte Projekt (Kapitel 3) wurde außerdem mit einem Smartphone entwickelt und getestet. Dabei handelt es sich um das Galaxy S i9000 der Firma Samsung, das mithilfe des Programms IP Webcam (Version 1.8.13) per kabellosem Netzwerk (wireless LAN) einen MJPEG Videostrom mit einer Auflösung von bis zu 800x480 Pixeln zugänglich macht. Die Anzahl Bilder pro Sekunde hängt dabei maßgeblich von der Leistung und aktuellen Auslastung des Smartphoneprozessors[8] ab (Pas, 2012).

2.4 Software

Für die Umsetzung der Arbeit mit der zuvor genannten Hardware kommen verschiedene Programmiersprachen und Entwicklungsinstrumente zum Einsatz, die im Folgenden genannt werden.

2.4.1 C++: OpenCV

OpenCV[9] ist eine frei verfügbare Bibliothek aus dem Bereich der Bildverarbeitung und des maschinellen Sehens. CV steht für englisch „Computer Vision". Die in C und C++ implementierte Bibliothek umfasst Algorithmen zum Lösen verschiedenster Aufgaben aus diesem Bereich. Sie wird hier in der Version 2.3 in C++-Projekte eingebunden, die mittels CMake[10] erstellt werden. Als Entwicklungsumgebung dient Microsoft Visual Studio 2010[11]. Vorrangig werden die bereitgestellten Funktionen für die Kamerakalibrierung, für die Berechnung räumlicher Transformationen und für den Umgang mit einem Videostrom genutzt.

[8] hier: Samsung Prozessor mit 1 GHz CPU-Taktrate
[9] http://sourceforge.net/projects/opencvlibrary/, abgerufen am 1.3.2012
[10] http://www.cmake.org/, abgerufen am 1.3.2012
[11] http://www.microsoft.com/germany/visualstudio/, abgerufen am 1.3.2012

2.4.2 Java: LeJOS

Im Auslieferungszustand befindet sich auf dem NXT-Baustein des LEGO Mindstorms NXT 2.0-Bausatzes eine Firmware bzw. ein Betriebssystem, das über eine mitgelieferte graphische Entwicklungsumgebung (NXT-G) von einem Computer aus programmiert wird. Damit können Einsteiger ohne programmiertechnische Vorbildung schnell und einfach relativ komplexe Robotik-Projekte umsetzen (Berns & Schmidt, 2010).

Neben weiteren graphischen Alternativen zu NXT-G, gibt es solche, die die Programmierung des NXT mittels objektorientierter Programmiersprachen oder Skriptsprachen ermöglichen. Eine Übersicht über die wichtigsten sowie deren Fähigkeiten geben Leimbach & Trella (2010) und Bredenfeld et al. (2010).

Im Rahmen dieser Arbeit wird leJOS[12] verwendet. Der Name leitet sich von dem spanischen Wort „lejos“ für „weit“ oder „entfernt“ her und steht für LEGO Java Operating System. Das Projekt bietet eine alternative Firmware für den NXT, die als angepasste Java Virtual Machine (JVM) dessen Programmierung mit Java ermöglicht, und eine zugehörige Klassenbibliothek, die sogenannte NXJ API (*Application Programming Interface*). Diese API bietet Zugriff auf die wichtigsten Java Klassen sowie auf spezielle Klassen zur Nutzung aller Funktionen des NXT, dessen Schnittstellen und diverser Sensoren und Aktuatoren. Hinzu kommen Klassen aus dem Bereich der Robotik. Eine PC API mit ähnlichem Umfang gibt die Möglichkeit, Berechnungen von dem in der Rechenkraft beschränkten NXT-Baustein auf einen wesentlich performanteren Computer auszulagern.

Als Entwicklungsumgebung, sowohl für Klassen auf dem NXT als auch für die der PC API, wird hier Eclipse[13] eingesetzt. Die Übertragung der Programme auf den NXT und die Kommunikation mit diesem erfolgt kabellos über Bluetooth, ist aber auch per Universal Serial Bus (USB) - Kabelverbindung möglich.

Nach Siciliano & Khatib (2008) sind graphische Programmierumgebungen auch in der Robotik von Vorteil, weil die schnell sehr komplex werdenden Abläufe abstrahiert werden können. Jedoch wurde im Rahmen dieser Abstraktion im Fall von NXT-G auf einige Dinge verzichtet, die für eine Arbeit, wie die vorliegende, benötigt werden. Bei der Firma LEGO handelt es sich um einen Spielzeughersteller. Vorteile von leJOS gegenüber NXT-G und anderen Möglichkeiten den NXT zu programmieren sind folgende:

- objektorientierte Sprache
- Plattformunabhängigkeit
- Threads
- multidimensionale Felder (Arrays)
- Rekursion
- Synchronisation (Threads)
- Ausnahme-/Fehler-Behandlung (Exceptions)
- Standard Java-Datentypen, wie float, long und String
- die meisten Klassen der Java-Pakete java.lang, java.util und java.io
- Robotik-API

Diese Vorteile sind gleichzeitig Argumente, leJOS in der vorliegenden Arbeit NXT-G vorzuziehen.

[12] http://lejos.sourceforge.net/nxj.php, abgerufen am 1.3.2012
[13] http://www.eclipse.org/, abgerufen am 1.3.2012

2.4.3 Matlab

Die Software Matlab[14] der Firma Mathworks kommt vorrangig bei der statistischen Analyse und Darstellung der gewonnenen Daten zum Einsatz. Die eigentliche Implementierung der Arbeit erfolgt mit den zuvor genannten Werkzeugen. Außerdem werden einzelne Algorithmen in Varianten in Matlab umgesetzt, getestet und verglichen.

[14] http://www.mathworks.de/products/matlab/, abgerufen am 1.3.2012

3 Umsetzung

Dieses Kapitel geht zunächst auf den Aufbau ein, der das im Rahmen der Arbeit umgesetzte Projekt darstellt. Dies beinhaltet insbesondere die Konzeption und den Aufbau des Roboters. Es folgt die Erläuterung der wesentlichen Schritte bis zu der Berechnung einer durch den Roboter erfassten Karte seiner Umgebung. Diese sind nach den dabei benötigten Koordinatensystemen gegliedert. Eine kurze Erläuterung verschiedener Kalibrierungsschritte und eine Übersicht über die programmiertechnische Umsetzung runden das Kapitel ab.

3.1 Aufbau

Die beiden wesentlichen Elemente des Aufbaus sind ein Roboter und eine Kamera, die diesen und seine Umgebung im Sichtfeld hat.

3.1.1 Kamera

Als Kamera kommen die in Kapitel 2.3.2 beschriebene PTZ-Netzwerk-Kamera beziehungsweise das Smartphone zu Testzwecken zum Einsatz. Die Kamera wird nahezu senkrecht (etwa 2,50 Meter) über der aufzunehmenden Szene nach unten blickend fest montiert. Eine Testumgebung, in der der Roboter agieren soll, wird auf dem Fußboden abgegrenzt und liegt vollständig im Kamerasichtfeld. Die Größe der Testumgebung beträgt eingeschränkt durch das maximale Sichtfeld der Kamera etwa 1,5 x 2 Meter. Grundsätzlich muss zwischen größerem Sichtfeld durch größere Entfernung der Kamera und dadurch bedingter schlechterer Sichtbarkeit des Roboters abgewogen werden: Je weiter weg der Roboter ist, also in je weniger Pixeln er abgebildet wird, desto größer muss seine Signalisierung sein, um erkannt zu werden. Es wird also angenommen, dass die Detektion bei gleichbleibender Signalisierungsgröße durch eine größere Entfernung zur Kamera schlechter wird. Eine Möglichkeit die maximal mögliche Entfernung des Roboters von der Kamera bei gleicher Signalisierungsgröße zu erhöhen, ist die Verwendung einer höheren Auflösung der Kamera, was jedoch den Rechenaufwand erhöht. Eventuelle Möglichkeiten einer skaleninvarianten Signalisierung werden nicht betrachtet.

Der Fußboden wird als eben angenommen und besteht aus Teppichfußboden. Die Beleuchtung erfolgt durch Tageslicht, durch die Raumbeleuchtung und eine auf die Szene gerichtete Lampe. Sowohl die gewählten Parameter für das Kamerabild und seine Übertragungsart und Qualität als auch die Ausrichtung und Zoomstufe der Kamera werden gespeichert und nach Unterbrechungen der Arbeit wieder hergestellt. Dabei wird angenommen, dass die gespeicherte Ausrichtung und Zoomstufe der Kamera durch ein Speichern und Wiederherstellen exakt rekonstruiert werden können, die Werte der Kamerakalibrierung für eine solche gespeicherte Konstellation also ihre Gültigkeit behalten.

3.1.2 Roboter

Bei dem eingesetzten Roboter handelt es sich um eine Eigenkonstruktion, die im Wesentlichen auf den Bauteilen eines Lego Mindstorms Baukastens basiert. Der Einteilung aus Kapitel 2.2.1 folgend, handelt es sich um einen Serviceroboter. Dies legen die im Rahmen der nachfolgend beschriebenen Konzeption des Roboters ermittelten Eckdaten dar.

3.1.2.1 Konzeption

Die Konzeption des Roboters folgt im Wesentlichen den von Boogarts et al. (2007) geschilderten Designschritten. Diese sind:

Schritt 1:
Inventur: Welche Bauteile mit welchen Fähigkeiten/Einschränkungen stehen zur Verfügung?

Schritt 2:
Sammeln von Ideen: Was tragen die einzelnen Teile an Funktionen bei?

Schritt 3:
Sammeln möglicher Aufgaben eines Roboters, mit den zur Verfügung stehenden Mitteln.

Schritt 4:
Abgleichen der Ideen mit den in Schritt 1 ermittelten Einschränkungen der Bauteile und gegebenenfalls Überarbeitung ausgehend von Schritt 2.

Schritt 5:
Design und Umsetzung eines Prototyps.

Schritt 6:
Testen und Verbessern dieses Prototyps in einer Testumgebung.

Schritt 7:
Schritte 2-6 wiederholen, um den Prototyp weiter zu entwickeln oder bisher nicht erkannte Einschränkungen aufzudecken, die der weiteren Entwicklung im Weg stehen.

Schritt 8 + 9:
Einsatz des entwickelten Roboters in einer realen Umgebung sowie Wartung des Roboters und Instandsetzung.

Bei Schritt 1 wird zwischen Bauteilen, also hier den Lego-Technik-Teilen, und „intelligenten“ Teilen wie Aktuatoren und Sensoren unterschieden. Tabelle 3.1 zeigt die Auflistung letzterer inklusive einiger direkt verfügbarer und programmiertechnisch umsetzbarer Funktionen (Schritt 2). Außerdem sind bekannte Einschränkungen und die Anzahl des jeweiligen zur Verfügung stehenden Teils aufgeführt. Im Allgemeinen ist die Frage nach den zur Verfügung stehenden Teilen eine wirtschaftliche. Je komplexer und hochwertiger die Teile, umso teurer können Anschaffung und Betrieb eines Roboters beispielsweise sein.

Tabelle 3.1: Auflistung der „intelligenten“ Teile und ihrer Funktionen

Teil	#	Funktionen	Programmierbare Funktionen	Bekannte/mögliche Einschränkungen
Motor	3	Antrieb einer Achse/eines Rades, Bewegung eines Arms	Vorgegebene Anzahl von Umdrehungen durchführen, Geschwindigkeit vorgeben	Leistung und Drehgeschwindigkeit beschränkt, sperriges Bauteil
Drehgeber (integriert)	3	Rotationsgeschwindigkeit und -richtung nachvollziehen	Zurückgelegte Distanz und aktuelle Geschwindigkeit ermitteln, Navigation per Odometrie	
Berührungssensor	2	Wahrnehmung eines Kontaktes mit einem Objekt (Kollision), Schalter	Zählen von Schaltvorgängen, Dauer eines Drucks ermitteln	Löst erst bei bestimmtem Druckgewicht aus
Ultraschallsensor	1	Distanzmessung, Bewegungserkennung	Hinderniserkennung, Objektverfolgung, Treppenstufen erkennen	Kegelförmige Signalausbreitung, Detektion abhängig von Größe und Struktur der Objekte, Übersprechen mit anderen Ultraschallquellen
Farbsensor	1	Farben und Helligkeit erkennen, farbig leuchten	Linie auf dem Untergrund verfolgen, Leuchten als Signalisierung nutzen	Nur auf kurze Distanz einsetzbar, geringe geometrische Auflösung, abhängig von Umgebungsbeleuchtung

Um auch Funktionen abzudecken, die erst mit mehreren dieser Teile möglich sind, werden außerdem Gruppen in der Konzeption eines Roboters berücksichtigt. Beispielsweise wäre eine Kombination aus Motor und Ultraschallsensor denkbar, die einen Rundumscan der Umgebung ermöglicht oder die Kombination von Ultraschall und Berührungssensor, um Hindernisse zu erkennen und falls dies nicht erfolgreich ist, die Kollision festzustellen, um beispielsweise die Motoren zu stoppen. Es können auch Kombinationen von Sensoren wünschenswert sein, die nicht ohne weiteres realisierbar sind. Ein Beispiel hierfür ist der Einsatz von zwei Ultraschallsensoren, um festzustellen, ob ein Hindernis rechts oder links vom Roboter liegt. Davon abgesehen, dass nur einer zur Verfügung steht, spricht das in Tabelle 3.1 aufgeführte Übersprechen dagegen.

Auf der Seite der Bauteile sind ähnliche Überlegungen nötig. Insbesondere fällt die Wahl des Fahrgestells an. Diese wird hier auf einen fahrenden Roboter beschränkt und wirft damit in erster Linie die Frage nach der Art und Anzahl an Rädern, welche von ihnen den Antrieb bilden und welche das Fahrzeug lenken, auf. Die Eignung der unterschiedlichen Möglichkeiten hängt vorrangig von der Art der geplanten Aufgaben und der Einsatzumgebung ab. Neben Boogarts et al. (2007) beschäftigen sich Siciliano & Khatib (2008) mit dieser Thematik. Roboter mit nur einem Rad (Prinzip eines Einrads) benötigen Mechanismen, um das Gleichgewicht zu halten; sie lassen sich nicht über reine Odometrie positionieren und kommen hier deshalb nicht in Frage.

Bei zweirädrigen kann zwischen zwei grundsätzlichen Ausführungen unterschieden werden: Die von einem Fahrrad bekannte Kombination eines festen und eines lenkbaren Rades ermöglicht zwar eine schmale Bauweise und benötigt nicht zwingend einen Mechanismus, der die Balance hält (je höher die Geschwindigkeit, desto besser). Sie hat aber den gravierenden Nachteil, dass ein Halten der Pose im Stillstand prinzipbedingt nicht ohne Weiteres möglich ist. Die zweite ist der in Kapitel 2.2.4 beschriebene Differentialantrieb mit zwei Rädern. Auch hier muss die Balance gehalten werden, es sei denn der Roboter wird so kompakt konstruiert, dass sein Schwerpunkt auf der Achse zwischen den beiden Rädern liegt.

Für dreirädrige Konstruktionen gibt es viele Möglichkeiten, die durch die Wahl unterschiedlicher Radkonstruktionen noch vielfältiger werden (Siciliano & Khatib, 2008). Der wichtigste Vorteil solcher Konstruktionen ist die Möglichkeit statisch stabile Modelle ohne Balanceproblematik zu bauen. Der simpelste Fall ist der von drei Rädern, der deshalb sehr häufig eingesetzt wird. Das am weitesten verbreitete Vorgehen ist dabei die Verwendung eines Differentialantriebs mit zwei Rädern und einem zusätzlichen Rad, das sich passiv mitdreht, lenkbar ist, nicht auf einer Achse mit diesen liegt und somit für ein stabiles Gleichgewicht sorgt. Vorteile sind unter anderem die einfache mechanische Struktur und Kinematik, die Möglichkeit einer Drehung auf der Stelle und die Möglichkeit systematische Fehler zu kalibrieren (siehe Abschnitt 3.6.3.1). Probleme bereiten unregelmäßige Oberflächen, da sich die Orientierung abrupt ändert, wenn eines der Antriebsräder den Bodenkontakt verliert (Siciliano & Khatib, 2008). Möglich ist auch die Verwendung des Differentialantriebs und einem stützenden Bauelement anstelle des dritten Rades, das auf dem Boden mitgleitet und so für Stabilität sorgt. Diese Lösung hat jedoch noch mehr Probleme mit unregelmäßigem Untergrund.

Kommt neben Rädern auch ein Raupenantrieb in Betracht, lassen sich die Vorteile eines zweirädrigen und die eines dreirädrigen Designs zum Teil kombinieren und die Nachteile gegeneinander aufwiegen. Zwei parallele „Ketten-Räder“, bestehend aus jeweils zwei verbundenen Rädern, ermöglichen den Einsatz als Differentialantrieb ohne drittes Rad: Das Gleichgewicht ist bereits durch die beiden Ketten hergestellt. Solange der so angetriebene Roboter nicht auf dem Untergrund aufsetzt, indem er beispielsweise seitlich auf eine Stufe rutscht, besteht prinzipbedingt Bodenkontakt zu beiden Ketten. Das Fahren auf unregelmäßigem Untergrund ist also besser möglich. Boogarts et al. (2007) führen zu diesem Kettenantrieb allerdings auch neue Nachteile auf. So rutschen die Ketten insbesondere bei der Kurvenfahrt durch gegensätzliches Drehen, was die Positionierung per Odometrie unsicher macht. Dies spiegelt sich auch in den Ergebnissen dieser Arbeit wider, die einen solchen Raupenantrieb beinhaltet, da seine Vorteile hier überwiegen (Kapitel 4).

Gemäß Schritt 3 (und 4) werden folgende Aufgaben für den im Rahmen dieser Arbeit konzipierten Roboter gesammelt:

- Bewegung auf unterschiedlichen Bodenbelägen wie Teppich oder Parkett
- Bewegung mittels Differentialantrieb zweier mit jeweils einem Motor angetriebener Ketten
- Erfassung und Speicherung der Umgebung in einer Karte
- Erfassung der Umgebung mittels Ultraschall- und Tastsensor
- Autonomes Abfahren der Umgebung unter Berücksichtigung möglicher Hindernisse
- Quasi kontinuierliche Ermittlung der Roboterpose zur Kartenerstellung
- Ermittlung der Pose per Odometrie und Korrektur anhand der Bildmessung mit einer Kamera
- Bildmessung anhand einer Signalisierung des Roboters

Damit wird deutlich, dass alle in Tabelle 3.1 gelisteten „intelligenten“ Teile mit Ausnahme des Lichtsensors in dem Schritt 5 folgend entworfenen Prototyp zum Einsatz kommen. Dieser könnte möglicher Weise zur Signalisierung für eine optische Zeitsynchronisation über das Kamerabild gemäß Kapitel 2.2.3 beitragen.

Für die Signalisierung des Roboters gegenüber der Kamera wird auf diesem eine ebene Tafel waagerecht angebracht, die die Form und Ausmaße des Roboters stark beeinflusst. Alle anderen Teile befinden sich unter ihr. Dabei ist der NXT-Baustein zentral zwischen den Antriebsketten montiert. Ein zur Stoßstange ausgebauter Tastsensor erkennt Kollisionen über die gesamte Front des Roboters. Kollisionen an den Seiten bei Drehungen oder beim Rückwärtsfahren hinten sind nicht erfassbar. Mit

dem Ziel, es überhaupt nicht zu Kollisionen kommen zu lassen, wird der Ultraschallsensor ebenfalls nach vorne geradeaus blickend horizontal (Sende- und Empfängerauge auf einer Höhe) angebracht. Er kann so Hindernisse erkennen, auf die der Roboter zu fährt. Die relative Position der Stoßstange und des Ultraschallsensors gegenüber dem Roboterzentrum wird als fest angenommen und kann deshalb in konstanten Werten ausgedrückt werden. Überlegungen, den dritten zur Verfügung stehenden Motor für eine rotierbare Anbringung des Ultraschallsensors einzusetzen, wurden verworfen. Der Roboter kann sich stattdessen aufgrund des Antriebs auf der Stelle drehen, falls eine Messung rundum gewünscht ist. Die Verwendung des Motors birgt theoretisch weitere Fehlerquellen, die sich in entsprechenden Tests ergaben: Die Stützen der Signalisierungstafel sind im Weg und die genaue Messrichtung relativ zu der Blickrichtung und -position des Roboters ist nicht mehr konstant, sondern unsicher.

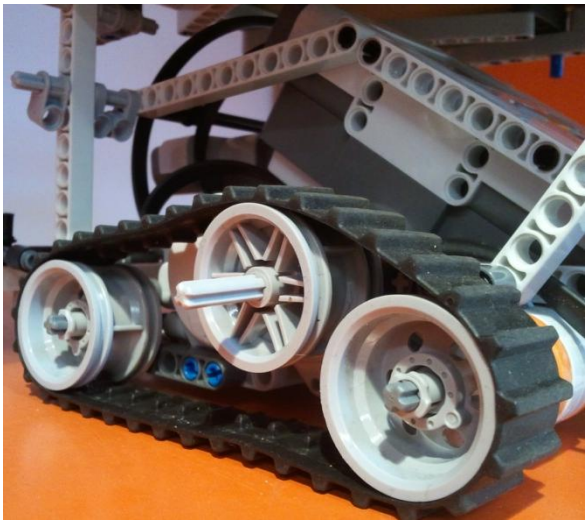

Abbildung 3.1: Kettenspanner

Abbildung 3.2: Klappbare Signalisierungstafel

Schritt 6 führt zu Korrekturen des Roboters: Ein drittes passives Rad wird als Kettenspanner eingebaut, um Unsicherheiten aufgrund durchrutschender und lockerer Ketten zu minimieren (Abbildung 3.1). Außerdem werden weitere Bauteile zur Stabilisierung des Robotergerüsts eingefügt, da sich Bewegungen innerhalb des Aufbaus ergaben, die der Annahme eines steifen in sich unbeweglichen Roboters entgegen stehen. Die Tafel für die Signalisierung wurde klappbar montiert, um die Bedienelemente des NXT-Bausteins für Wartungszwecke erreichen zu können (Abbildung 3.2).

Den fertigen Prototyp zeigen Abbildung 3.3 und Abbildung 3.4.

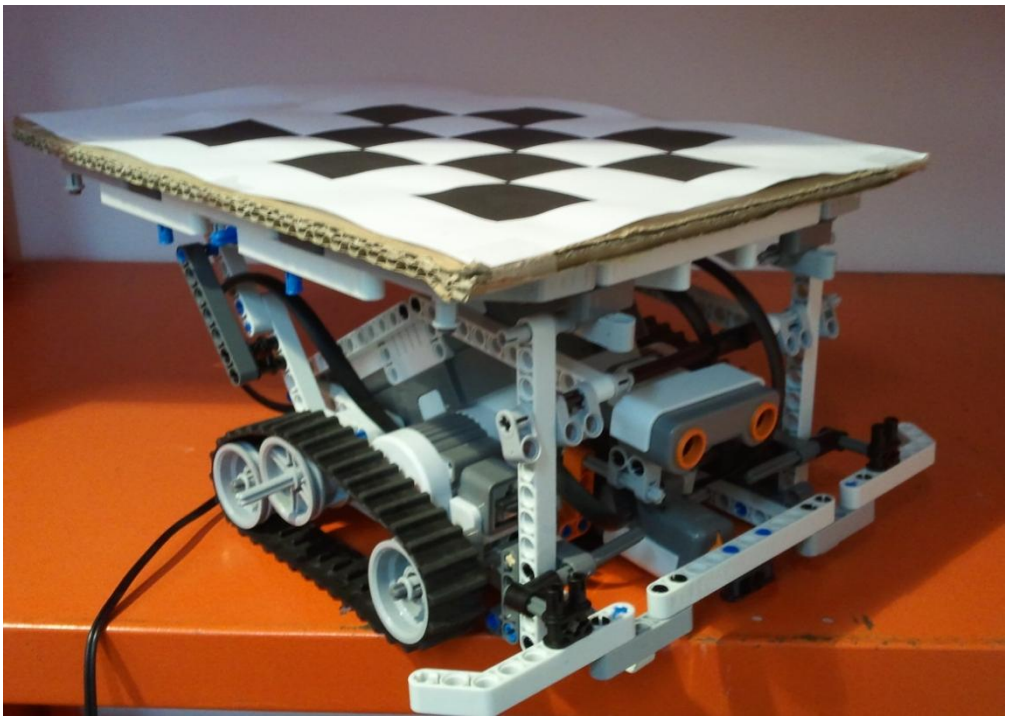

Abbildung 3.3: Der fertige Prototyp des Roboters

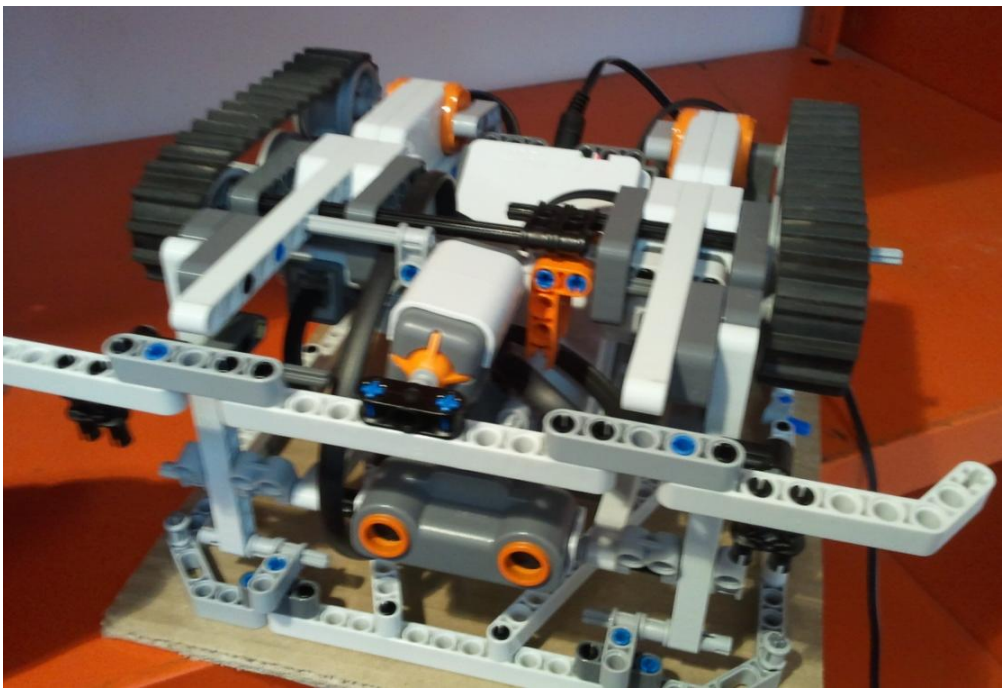

Abbildung 3.4: Der fertige Prototyp des Roboters von Unten

Als (2D-)Zentrum des Roboters, auf das sich dessen Pose bezieht, wird seine Drehachse definiert. Sie ergibt sich, wenn der Roboter sich durch gegenläufiges Drehen der Ketten mit gleicher Geschwindigkeit auf der Stelle im Kreis dreht. Unter der Annahme, dass die Ebene der Signalisierung parallel zur Bodenebene auf dem Roboter angebracht ist, ist diese Achse orthogonal zu diesen beiden Ebenen. Da im Folgenden nur die zweidimensionale Pose des Roboters bezogen auf diese Ebenen benötigt wird, muss das dreidimensionale Zentrum des Roboters auf dieser Achse nicht definiert sein.

3.1.3 Messanordnung

Der Versuchsaufbau (Abbildung 3.5) umfasst, neben der gemäß Abschnitt 3.1.1 montierten Kamera und dem zuvor beschriebenen Roboter, die durch das Kamerasichtfeld in ihren Ausmaßen eingeschränkte Testumgebung und einen Personal Computer (PC). Die Testumgebung ist ein auf ebenem Teppichfußboden durch Pappkartons, Wand, Holzbretter und eine Metallplatte abgegrenzter Bereich (Abbildung 3.6). Die Begrenzung folgt dabei nur zum Teil den durch den Bildausschnitt der Kamera gegebenen Grenzen: Die Ecken sind zum Teil deutlich abgerundet.

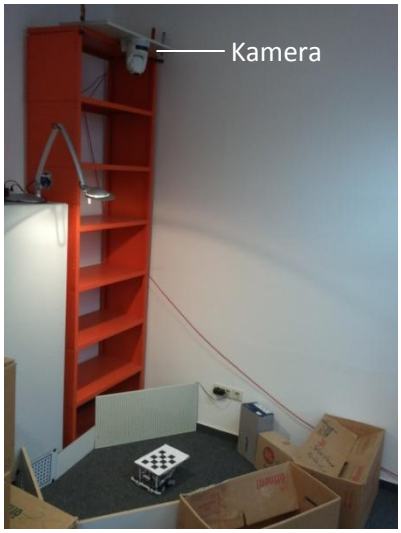

Abbildung 3.5: Messanordnung (ohne PC)

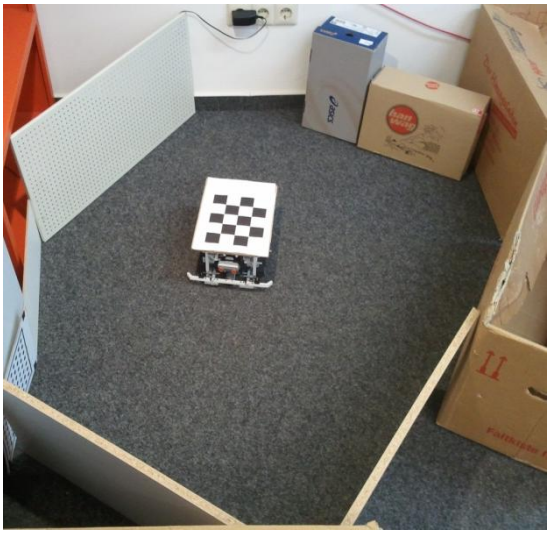

Abbildung 3.6: Testumgebung

Die Kommunikation zwischen Computer und Netzwerkkamera erfolgt per kabelgebundenem IP-Netzwerk, die zwischen Computer und NXT-Baustein per Bluetooth. Zunächst erfolgte die Kommunikation hier per USB. Die geringere Datenübertragungsrate von Bluetooth gegenüber USB ist für die hiesige Anwendung ausreichend, so dass der Vorteil, auf ein Kabel verzichten zu können, überwiegt. Der Roboter kann sich so frei bewegen und drehen, ohne dass ein Kabel mitgeführt werden muss.

3.2 Fahrstrategie

Der an einer beliebigen Position in der Testumgebung platzierte Roboter soll diese mit möglichst kurzen Wegen und möglichst detailliert aufnehmen. Die Ausgangssituation ist die, dass keinerlei Informationen über die Umwelt existieren. Damit ist eine Planung des zu fahrenden Weges nicht möglich und die Fahrstrategie ist zunächst darauf beschränkt, die unbekannte Umgebung zu erfassen. Dies geschieht über die Stoßstange als Tastsensor und den Ultraschallsensor. Der Roboter muss sich fortbewegen, um die Umgebung zu erkunden und nach Möglichkeit alle Bereiche mittels Ultraschall anzuzielen. Die einfachste Strategie ist dabei, ihn geradeaus fahren zu lassen und die Richtung zu ändern, wenn die Kollision mit einem Hindernis droht. Dies ist der Fall, wenn die gemessene Distanz einen vorzugebenden Wert unterschreitet. Die Ultraschallmessung muss während der Geradeausfahrt also kontinuierlich erfolgen, was mithilfe des entsprechenden Messmodus umgesetzt wird. Versagt der Ultraschallsensor bei der Erkennung eines Hindernisses und es kommt zu einer Kollision, detektiert dies der Tastsensor an der Stoßstange. In diesem Fall muss der Roboter vor der Richtungsänderung zurücksetzen, um sich drehen zu können. Abbildung 3.7 zeigt diese Strategie. Wird nach der Drehung wieder ein Hindernis erkannt, wird die Geradeausfahrt nicht ausgeführt und wieder ausgewichen. Die Drehung erfolgt per Zufall nach rechts oder links um einen zufälligen Winkel in einem festzulegenden Gradbereich. Der Zufall für die Drehrichtung wird jedoch eingeschränkt: Um

ein länger andauerndes Drehen des Roboters in Ecken zu vermeiden, wird die Richtung der jeweils letzten beiden Drehungen gespeichert und ausgewertet und damit verhindert, dass nach einer Drehung nach links und einer nach rechts wieder links herum gedreht wird und umgekehrt.

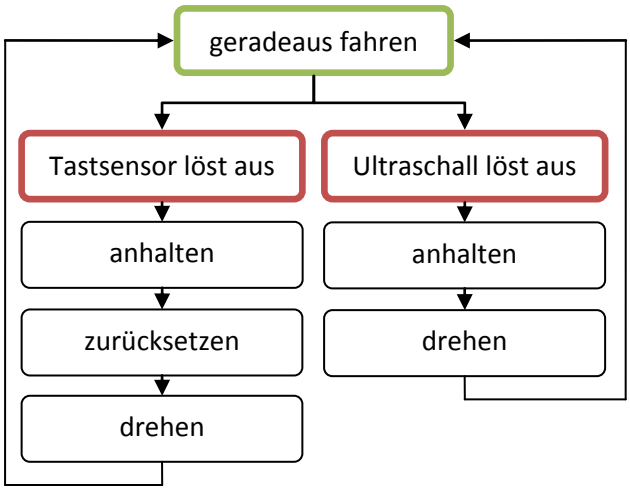

Abbildung 3.7: Fahrstrategie
So lange weder Ultraschall noch Tastsensor auslösen, wird geradeaus gefahren. Sonst wird angehalten und auf das detektierte Hindernis reagiert, bevor die Geradeausfahrt wieder aufgenommen werden kann.

Festzulegende Werte dieser Strategie sind:

- Fahrgeschwindigkeit
- Drehgeschwindigkeit
- minimal zugelassene Hindernisentfernung
- maximal und minimaler zugelassen Drehwinkel
- Entfernung, die im Fall einer Kollision zurückgesetzt wird

Die Einschränkung der zufälligen Drehrichtung ist ein erster Schritt, das Verfahren effizienter zu gestalten. Diese Vorgehensweise benötigt unter Umständen sehr viele Wege und damit viel Zeit, um die Umgebung zu erfassen, da beispielsweise nicht berücksichtigt wird, welche Teile der Karte schon abgefahren wurden und welche nicht. Es gibt Bereiche, die viel häufiger als andere angefahren werden, was die Dichte der Messungen inhomogen und schwer abschätzbar macht.

Möglichkeiten für bessere Verfahren finden sich beispielsweise in (Siciliano & Khatib, 2008).

3.3 Signalisierung für das Tracking der Roboterplattform

Die passive Signalisierung für das Tracking der Roboterplattform per Kamera bildet ein planares Muster. Aktive Signalisierungsmöglichkeiten beispielsweise über Lichter werden nicht verfolgt. Bekannt aus der Kamerakalibrierung mit OpenCV ist ein Schachbrettmuster, welches eigenen Tests zufolge gut geeignet ist, ein lokales Koordinatensystem der Ecken zu definieren und eine automatische Erkennung zu begünstigen.

Eine Alternative wären beispielsweise in gleichen Abständen rasterförmig angeordnete Kreise. Hier ist allerdings die Festlegung eines Ursprungs für das lokale System der Signalisierung zweideutig. Denkbar ist auch die Nutzung aus der Photogrammetrie bekannter Zielmarken (diese beschreibt (Luhmann, 2010)).

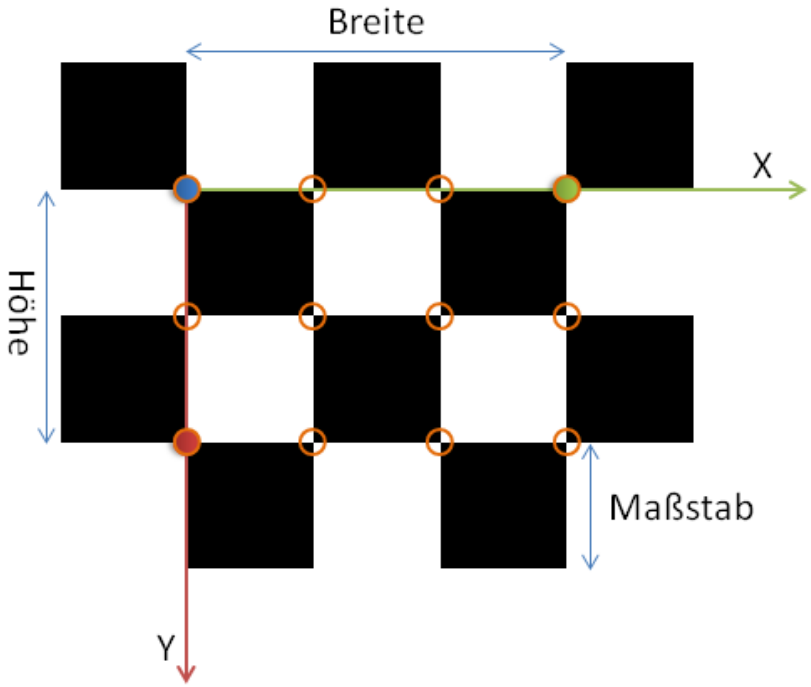

Abbildung 3.8: Definition der Schachbrettkoordinaten
Gezeigt ist ein 4x3 großes Schachbrettmuster bestehend aus gleichgroßen Quadraten, deren Kantenlänge den Maßstab des Musters darstellt. Die Größenangabe bezieht sich auf die genutzten Punkte, die Ecken aneinander grenzender Quadrate (orange umkreist). Ursprung des Koordinatensystems ist der blaue Punkt, der sich durch die Anordnung der Schwarz-Weiß-Muster von dem schräg gegenüberliegenden eindeutig unterscheiden lässt (Das Eckquadrat ist dort weiß und nicht schwarz). Die X-Richtung definiert der Eckpunkt (grün) entlang der längeren Seite (Breite). Y definiert der entsprechende Punkt (rot) entlang der Höhe.

Die Definition eines lokalen Koordinatensystems für ein Schachbrettmuster zeigt Abbildung 3.8 am Beispiel eines 4x3 großen Musters. Die Größenangabe entspricht der Anzahl verfügbarer Punkte. Breite und Höhe nach der Anzahl von Quadraten wären jeweils um eins erhöht.

Die drei Variablen, die ein solches Schachbrettmuster charakterisieren sind dessen Breite, Höhe und die Kantenlänge der Quadrate (Maßstab).

Die mindestens benötigte Größe des Musters kann von der Anzahl an zu lösenden Freiheitsgraden abhängig gemacht werden. Je größer das Muster ist, desto mehr Punkte stehen jedoch im Sinne einer Redundanz zur Verfügung und dienen der Ausgleichung der Parameter. Ändern sich die Ausmaße der Mustertafel dabei nicht, werden die Schachbrettquadrate (Maßstab in Abbildung 3.8) mit steigender Anzahl jedoch kleiner. Je weiter das Schachbrett von der Kamera entfernt ist, desto kleiner werden ebenfalls die Quadrate im aufgenommenen Bild. Je nach geometrischer Auflösung der Kamera werden die Schachbrettecken ab einer bestimmten Entfernung nicht mehr erkennbar sein. Der Vergrößerung des Musters zugunsten einer Überbestimmung steht also die damit beschränkte Entfernung von der Kamera gegenüber.

Da eine längere und eine kürzere Ausdehnung des Schachbrettmusters sowie eine eindeutige Zuordnung des Ursprungs unabdingbar sind, beträgt die minimale Größe 3x2. 2x1 legt die Y-Richtung nicht mehr fest, 2x2 würde gleiche Höhe und Breite bedeuten.

Aussagen über die mindestens benötigte Schachbrettgröße für die Kamerakalibrierung und das Tracking finden sich in den entsprechenden Abschnitten 3.6.1 und 3.4.1, Aussagen über die Mindestgröße der Quadrate und damit deren maximale Anzahl für den getesteten Aufbau sowie Angaben dazu, wie schräg die Sicht auf ein solches Schachbrettmuster sein kann, in den Ergebnissen in Kapitel 4.

3.4 Ermittlung der Roboterpose

3.4.1 Bild- und Schachbrettkoordinaten

Ziel ist nun, die Pose des Roboters in Bezug zu einem Objektkoordinatensystem zu jedem Zeitpunkt zu bestimmen. Dies ist zunächst das zu diesem Zweck um eine weitere Dimension erweiterte Schachbrettkoordinatensystem: Es wird eine aus der Schachbrettebene heraus ragende Z-Achse hinzugefügt. Die Schachbrettpunkte haben damit alle die Z-Koordinate 0. Mit den so gewonnenen 3D-Punkten können aus einem einzigen Kamerabild mithilfe der Bildkoordinaten dieser Schachbrettpunkte die räumliche Ausrichtung und die Entfernung des Schachbretts von der Kamera berechnet werden. Dies geschieht über 3D-2D-Punktkorrespondenzen: Jeder Schachbrettpunkt liegt in 3D-Schachbrettkoordinaten und in 2D-Bildkoordinaten vor. Das Ergebnis eines damit durchgeführten Rückwärtsschnitts sind die 6 Parameter der äußeren Orientierung der Kamera bezogen auf das Schachbrettkoordinatensystem.

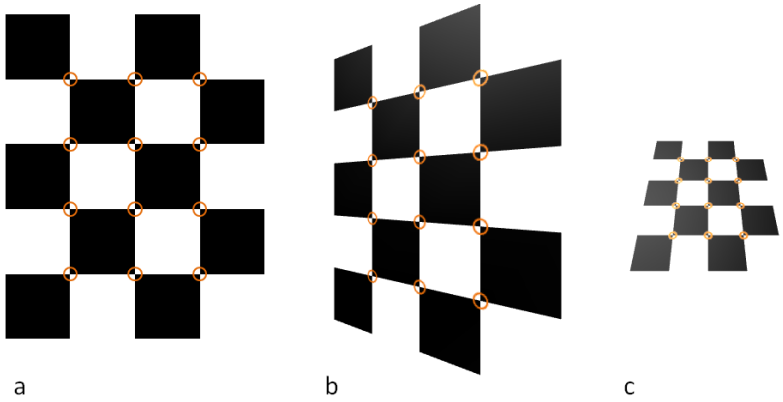

Abbildung 3.9: Schachbrettansichten
a) Aufsicht b) entlang der längeren Seite gedreht c) entlang der kürzeren Seite gedreht und Entfernung vergrößert

Alternativ lässt sich vereinfachend aus 2D-2D-Punktkorrespondenzen direkt eine Homographie (Kapitel 2.1.3.2, Formel (2.15)) zwischen den Bildkoordinaten in der Bildebene und den Schachbrettkoordinaten in der Ebene des Schachbrettes berechnen, die die Schachbrettebene auf die Bildebene abbildet und umgekehrt (3.1).

$$x^{Schach} = H_{Bild}^{Schach} x^{Bild} \tag{3.1}$$

Dies stellt eine Vereinfachung dar, da der Umstand ausgenutzt wird, dass alle Objektpunkte in einer Ebene liegen. Für die Bestimmung der acht Freiheitsgrade der Homographie wären vier Schachbrettecken à zwei Koordinaten ausreichend. Die minimale Schachbrettgröße wäre also 2x2. Diese entspricht jedoch nicht den Anforderungen aus Abschnitt 3.3, weshalb die minimale Schachbrettgröße für das Tracking bei 3x2 liegt.

Die Homographie berücksichtigt mit ihren acht Freiheitsgraden Scherungen der Koordinatenachsen. Da das Schachbrett aus Quadern besteht, würden hier streng genommen wie bei den zuvor genannten 3D-2D-Punktkorrespondenzen auch sechs Freiheitsgrade ausreichen. Dennoch wird eine Homographie genutzt, da deren Schätzung in OpenCV bereits mitgeliefert ist.

Abbildung 3.9 zeigt beispielhaft drei verschiedene Ansichten eines Schachbretts mit unterschiedlicher Ausrichtung und Entfernung zum Betrachter. Sie verdeutlicht damit anschaulich das Prinzip, mit dem sich die Berechnung der dreidimensionalen Position ohne Stereobetrachtung vorstellen lässt: Die Abbildung des Schachbretts ist so verzerrt, dass dichter liegende Punkte einen größeren Abstand

voneinander haben, weiter entfernt liegende dementsprechend einen kleineren. Da der wahre Abstand aber bekannt ist, lässt sich aus der Abbildung die Pose des Schachbretts ableiten.

3.4.2 Das Koordinatensystem der Kamera

Um nun das Tracking über die Zeit in Videobildern zu realisieren, muss die Pose in Objektkoordinaten nach Möglichkeit in jedem Einzelbild des Videos aktualisiert werden. Bewegt sich der Roboter, ändert sich jedoch das zuvor anhand des Schachbretts definierte Objektkoordinatensystem, da es mit bewegt wird. Die Pose in diesem ändert sich nicht – dafür aber relativ dazu die der Kamera. Da die Kamera nach wie vor unbewegt ist und sich der Roboter bewegt, wird ein dreidimensionales kamerafestes Koordinatensystem definiert. In diesem werden die anhand des Schachbretts definierten Koordinatensysteme aller Zeitpunkte zusammengeführt, was den bewegten Roboter korrekt wieder gibt und der unbewegten Kamera Rechnung trägt. Das erste (Epoche 0) der Systeme wird als dieses Objektkoordinatensystem $(X^O_{(0)}, Y^O_{(0)}, Z^O_{(0)})$ festgehalten. Alle weiteren gemessenen Roboterposen werden über das kamerafeste System in dieses System der Signalisierung übertragen. Dies zeigt Abbildung 3.10.

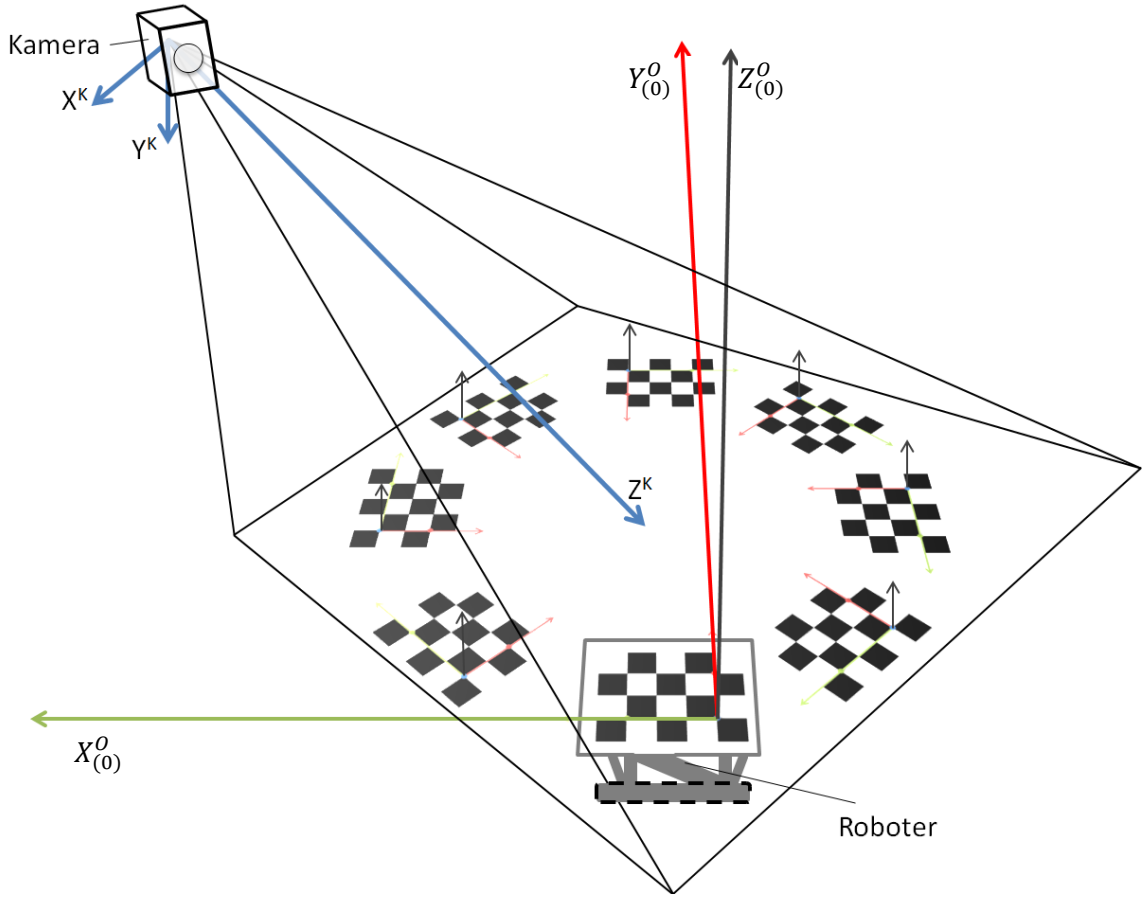

Abbildung 3.10: Das Kamera- und das Objektkoordinatensystem
Gezeigt sind das in der Kamera gelagerte Kamerakoordinatensystem (X^K, Y^K, Z^K) sowie einige Messungen des Schachbretts bei einer Kreisfahrt des Roboters innerhalb des Kamerabildes. Das Objektkoordinatensystem bildet das um die Z-Achse ergänzte Schachbrettkoordinatensystem (vgl. Abbildung 3.8, grüne und rote Achse, $X^O_{(0)}, Y^O_{(0)}, Z^O_{(0)}$) zu Epoche 0, die hier mit dem Roboter skizziert ist. Die lokalen Schachbrettkoordinatensysteme für jede weitere Epoche sind ebenfalls angedeutet.

Das dreidimensionale Kamerakoordinatensystem (X^K, Y^K, Z^K) hat seinen Ursprung in dem Projektionszentrum der Kamera. Eine Nahaufnahme in Form einer schematischen Skizze des Kamerainneren

für die Definition des Kamerakoordinatensystems zeigt Abbildung 3.11. Die Z^K-Achse steht senkrecht auf der Bildebene und geht in Blickrichtung. Y^K verläuft im Bild nach unten, X^K komplettiert ein rechtshändiges System und geht nach rechts. Die Ausrichtung von X^K und Y^K entspricht damit der des Bildkoordinatensystems (x^B, y^B) für den Fall, dass das Bild vor dem Projektionszentrum gedacht ist, so dass der Ursprung der Bildkoordinaten in der oberen linken Ecke des Bildes liegt. Die auf der Bildebene abgebildeten Achsen des Kamerakoordinatensystems sind gestrichelt eingezeichnet.

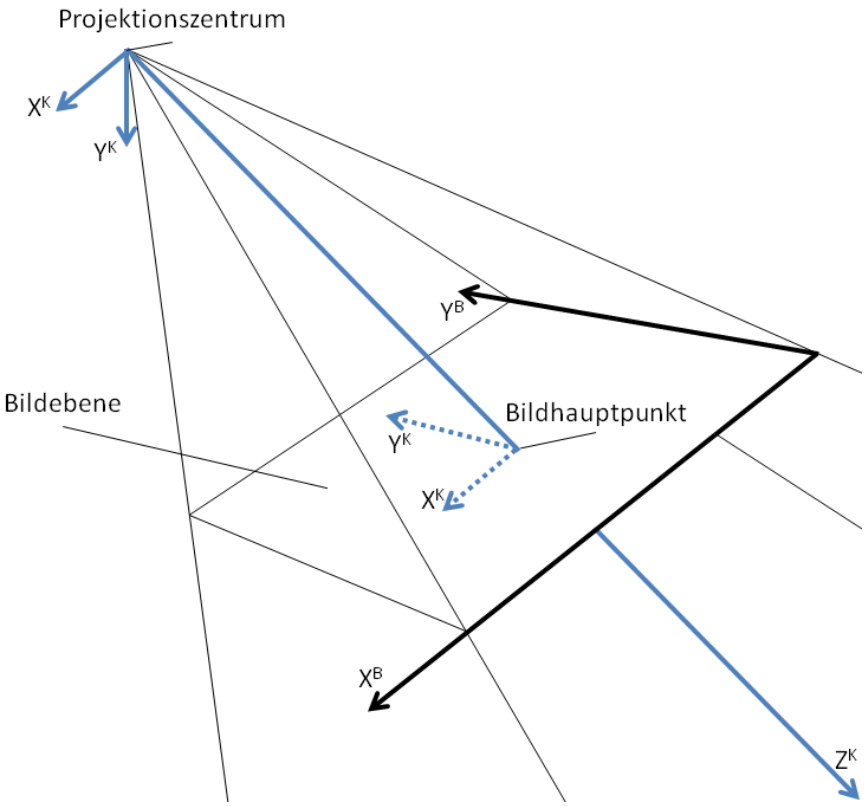

Abbildung 3.11: Das Kamera- und das Bildkoordinatensystem
Schematische Detailansicht des Kamera- (X^K, Y^K, Z^K) und des in der Bildebene liegenden Bildkoordinatensystems (x^B, y^B) mit Projektionszentrum und Bildhauptpunkt

3.4.3 Das Koordinatensystem der Roboterplattform

Die in Abschnitt 3.1.2 getroffene Annahme, das Schachbrett sei parallel zu einem ebenen Boden, auf dem sich der Roboter bewegt, wird nun für die Definition eines Roboterkoordinatensystems genutzt. Denn sie bedeutet, dass die Schachbrettebenen aller Epochen parallel sind, solange der Roboter sich auf einem ebenen Boden bewegt. Aus diesem Grund kann vereinfachend auf eine zweidimensionale Betrachtung zurückgegangen werden, das 2D-Roboterkoordinatensystem sei parallel zu der X-Y-Ebene des Objektkoordinatensystems.

Um nun die gesuchte Pose des Roboters aus der des Schachbretts abzuleiten, also zwischen dem Objektkoordinatensystem und dem Koordinatensystem der Roboterplattform transformieren zu können, müssen sein Drehzentrum in 2D-Objektkoordinaten (x_{Rot}, y_{Rot}) und seine Ausrichtung relativ zu diesem bekannt sein. Abbildung 3.12 zeigt den Zusammenhang. Die Kalibrierung des Zentrums erläutert Abschnitt 3.6.2; die Ausrichtung wird als konstruktionsbedingt fest angenommen. Die Fahrtrichtung verläuft entgegen der X-Achse $(X^O_{(0)})$ des Schachbretts.

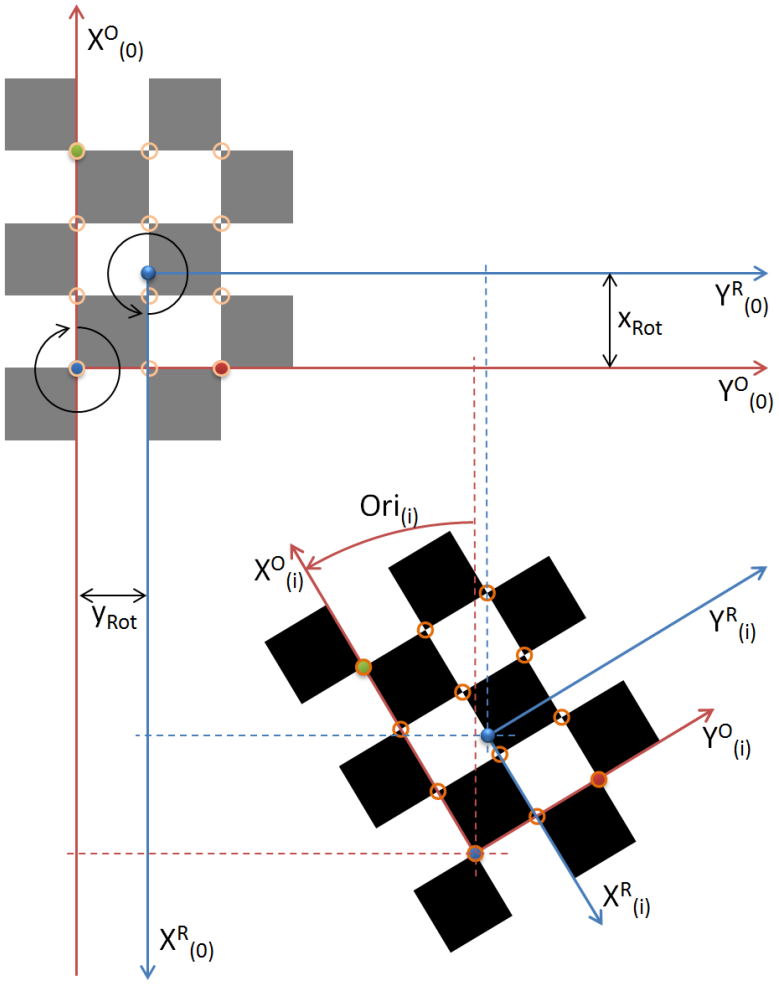

Abbildung 3.12: Ebene Koordinatensysteme von Roboter und Signalisierung
Gezeigt ist das Schachbrett zur Nullepoche (grau) sowie eines zur Epoche i nach einer Bewegung des Roboters. Ersteres definiert das hier nur zweidimensional betrachtete Objektkoordinatensystem (rot, $X^O_{(0)}, Y^O_{(0)}$). In diesem liegt das (2D)-Rotationszentrum des Roboters vor, das als Ursprung des Roboterkoordinatensystems (blau, $X^R_{(0)}, Y^R_{(0)}$) fungiert. x_{Rot} und y_{Rot} sind dabei die Koordinaten des Zentrums in Objektkoordinaten. Winkel (schwarz) in beiden Systemen beginnen von der X-Achse und drehen in Richtung der Y-Achse.
Epoche i repräsentieren entsprechend die beiden lokalen Systeme $X^O_{(i)}, Y^O_{(i)}$ und $X^R_{(i)}, Y^R_{(i)}$. Dabei wird zunächst die Verbindung innerhalb des Schachbrettsystems (rot gestrichelt) hergestellt, womit dann auch die Transformation in übergeordnete Roboterkoordinaten möglich ist (blau gestrichelt). Die Orientierung $Ori_{(i)}$ des Roboters muss dabei unter Berücksichtigung der unterschiedlichen Drehrichtungen übertragen werden.

Der Ursprung des Roboterkoordinatensystems ist das Drehzentrum. Die X-Achse ($X^R_{(0)}$) verläuft in Fahrtrichtung geradeaus des Roboters, also entgegen $X^O_{(0)}$ und parallel dazu. Die Y-Richtungen sind gleichgerichtet und parallel.

Liegt die initial gemessene Epoche 0 vor und sind Objekt- und Roboterkoordinatensystem festgelegt, kann die aktuelle Pose des Roboters zu einer Epoche i in Objektkoordinaten berechnet werden. Die Transformation nutzt dabei die feste Position der Kamera und die zuvor getroffene Vereinfachung paralleler Schachbrett- und Bodenebene aus. Die Transformation des lokalen Schachbrettsystems ($X^O_{(i)}, Y^O_{(i)}$) in das festgehaltene ($X^O_{(0)}, Y^O_{(0)}$) reduziert sich dadurch auf die Abbildung von Ebenen, gemäß Abschnitt 3.4.1 mithilfe einer Homographie. Diese wird für die Epoche 0 berechnet und auf alle weiteren Epochen i als Transformation (Formel (3.1)) angewendet.

Die mit dem Schachbrettsystem fest verknüpfte Lage des Robotersystems wird abschließend genutzt, um dessen Pose in Epoche i zu berechnen: Ausgehend von den per Homographie berechneten Ob-

jektkoordinaten des Schachbrettursprungs ergeben sich die des Roboterzentrums über eine Drehung um $Ori_{(i)}$ und Translationen mit x_{Rot} und y_{Rot} mittels polarem Anhängen. Die Umrechnung der Roboterpose in das Robotersystem entspricht dann lediglich noch der Umkehrung der X-Achse, Berücksichtigung der Offsets (x_{Rot}, y_{Rot}) und der unterschiedlichen Winkeldefinitionen beider Systeme.

3.5 Integration der Beobachtungen

3.5.1 Synchronisation

Die Pose des Roboters wird nicht nur per Bildmessung, sondern per Odometrie auch durch diesen selbst ermittelt. Die Kombination der Beobachtungen des Videosystems und des Roboters macht ein Synchronisieren der Zeit zwischen beiden Systemen nötig. Von den in Abschnitt 2.2.3 genannten Möglichkeiten wird die Verwendung von Zeitstempeln in einer ereignisorientierten Vorgehensweise implementiert: Jeder Bildmessung sowie jeder Messung eines Sensors des Roboters wird ein Zeitstempel hinzugefügt. Kamera, PC, sowie NXT besitzen eine Uhr, die dazu herangezogen wird. Der nötige kontinuierliche Abgleich zwischen Kamera und PC erfolgt über einen NTP-Server innerhalb des Netzwerks.

Die Uhr des NXT zählt mit jedem Einschalten von 0 beginnend Millisekunden. Der Zeitabgleich wird hier per Bluetooth über die PC-Uhr realisiert: Der PC schickt alle 500 ms seine aktuelle Zeit an den NXT, der mit dieser und seiner eigenen antwortet. Die dafür benötigte Zeit lässt sich durch ein erneutes Abspeichern der PC-Zeit, beim Empfang der NXT-Zeit abschätzen. Sie beträgt wenige Millisekunden.

Beide Zeitskalen lassen sich durch einen Offset verknüpfen. Dieser wird nach dem Aufbau einer Bluetoothverbindung aus über 30 Sekunden gesammelten Zeitstempelpaaren gemittelt. Um eine mögliche Drift der NXT-Uhr abzufangen, wird dieser Vorgang ständig wiederholt. Drift von PC- oder Kamera-Uhr werden über den minütlichen NTP-Abgleich minimiert.

Die Zeitstempel des NXT werden zusammen mit dem zugehörigen Messwert an den PC übertragen, der sie anhand des ermittelten Offsets in die NTP-Zeit umrechnet. Die Bilder der Kamera enthalten den Zeitstempel der Bildaufnahme direkt im Header der JPEG-Einzelbilder des MJPEG-Videostroms.

3.5.2 Korrektur der Pose per Kamera

Die Datenrate, in der die Pose aus der Odometrie und die von der Kamera vorliegen, variiert aufgrund verschiedenster Faktoren. Beispielsweise dauert die Detektion des Schachbretts je nach dessen Ansicht unterschiedlich lange und um die Bluetooth-Übertragung konkurrieren verschiedene Übertragungspakete (kontinuierlicher Zeitabgleich, mehrere Sensoren). Im Allgemeinen wird die Datenrate der Odometrie jedoch über der der Kamera liegen, so dass auf mehrere Odometrie-Posen eine per Kamera ermittelte kommt.

Sowohl die von der Kamera gelieferte als auch die per Odometrie ermittelte Roboterpose werden auf dem PC weiter verarbeitet. Die Korrektur beziehungsweise Kombination der beiden erfolgt also dort. Dabei werden die unterschiedlichen Fehlercharakteristiken ausgenutzt: Die Odometrie liefert hochfrequentere Daten. Die Fehler zwischen den Epochen sind gering, summieren sich über die Zeit aber auf (Drift). Die Kamera liefert hingegen in niedrigerer Frequenz Messungen, deren Genauigkeit mit der Zeit nicht schlechter wird. Diese dienen der Odometrie als Stützstellen, indem sie den auflaufenden Fehler zurücksetzen. Da dies auf dem PC stattfindet, kann auch erst dort, wie im nächsten Ab-

schnitt erläutert, die Kartierung stattfinden, die für die Integration der Messwerte zunächst die Roboterpose benötigt.

Durch Fehler oder Verzögerungen in der Datenübertragung und insbesondere Probleme bei der Detektion des Schachbretts (im Extremfall dessen Verdeckung) können jedoch für beide Messmethoden Datenlücken auftreten, die behandelt werden müssen. Außerdem können Messungen unterschiedlich verzögert am PC eintreffen. Da die Datenübertragung anhand von Warteschlangen implementiert wird, kann sich die Reihenfolge der Daten jedoch nicht ändern. Sie liegen also sicher chronologisch vor.

Diese Rahmenbedingungen ermöglichen das in Abbildung 3.13 gezeigte Vorgehen bei der Kombination beider Quellen für die Roboterpose. Empfangene Odometrie-Posen werden in einer Warteschleife gesammelt, bis eine neue der Kamera eintrifft. Alle Posen der Warteschleife deren Zeitstempel angibt, dass sie vor der nun empfangenen Kamera-Pose liegen, werden der Warteschleife entnommen: Sie liegen zwischen zwei Kamera-Posen. Liegen bereits aktuellere Posen der Odometrie vor, verbleiben diese in der Warteschleife. Vereinfachend wird angenommen, dass Anfang und Ende der entnommenen Reihe von Posen jeweils der aktuellen und der zurückliegenden Kamera-Pose entsprechen, also denselben Zeitstempel besitzen. Da die Messungen nicht synchron stattfinden, unterscheiden sich die Zeitstempel maximal um die aktuelle Messrate der Odometrie. Da die Bewegung des Roboters in dieser kurzen Zeit minimal ist, wird dies vernachlässigt. Der bisher angefallene Fehler entspricht der Differenz der letzten Posen; der zwischen beiden Kamera-Posen hinzugekommene, der Differenz der aktuellen Posen.

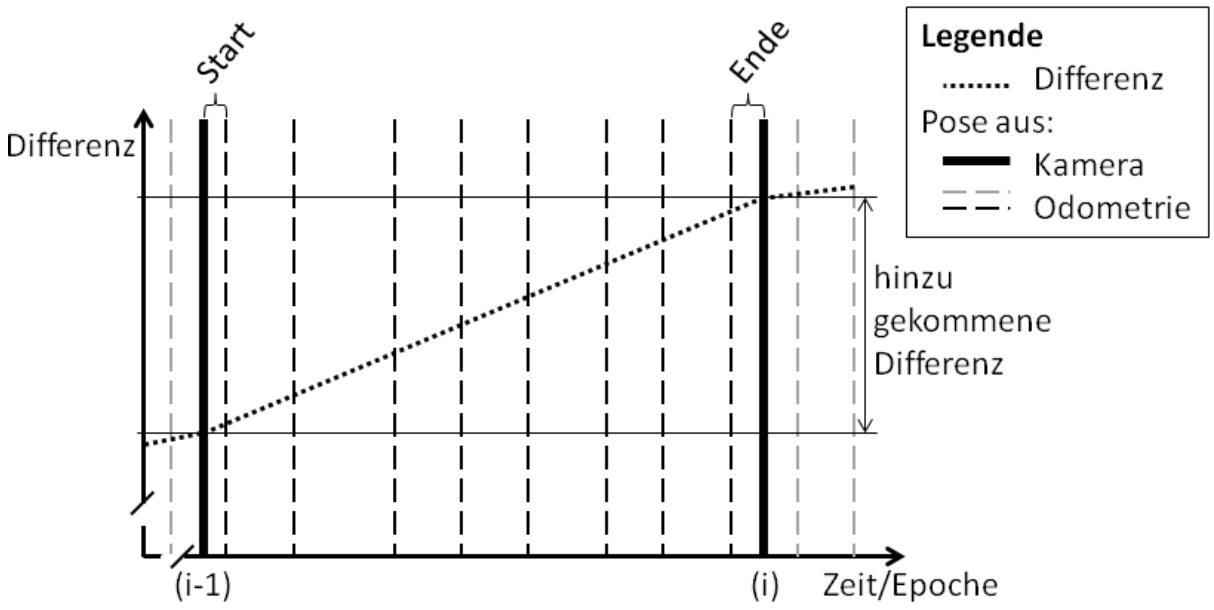

Abbildung 3.13: Korrektur der Posen aus der Odometrie
Schematische Darstellung zweier durch die Kamera gemessener Posen auf einer Zeitachse (Epoche i und i-1, fett) und der dazwischen liegenden Odometrie-Posen (schwarz gestrichelt). Ältere wurden bereits zwischen Epoche i-1 und i-2 korrigiert, neuere warten auf die Epoche i+1 (grau gestrichelt). Die jeweils herrschende Differenz ist ebenfalls gezeichnet (gepunktet, vertikale Achse).

Wird vereinfachend von einer gleichförmigen Bewegung des Roboters zwischen den Zeitpunkten ausgegangen, kann der Fehler für die dazwischen liegenden Odometrie-Messungen linear interpoliert werden. Als Minimallösung umgesetzt ist hier das Aufaddieren des jeweils aktuellen Fehlers, der positiv oder negativ sein kann, auf die jeweils zwischen zwei Kamera-Messungen liegenden Odometrie-Posen. Dies bedeutet, dass die Odometrie mit jeder Kamera-Pose zurückgesetzt wird, diese Pose

also als fehlerfreie einfließt. Diese wiederum getroffene Vereinfachung liegt in der vielfach besseren Genauigkeit der Kamera-Pose gegenüber der Odometrie begründet. Gleichzeitig fließen damit aber auch Ausreißer direkt in die Pose ein: Es findet keine Gewichtung beider Verfahren statt.

Für eine strenge Abschätzung der Genauigkeit und der Zuverlässigkeit der resultierenden Roboterpose und eine statistisch robuste Kombination beider Messmethoden ist eine Verfeinerung der hier gezeigten Vorgehensweise beispielsweise mittels Kalman-Filterung (Niemeier, 2002) denkbar.

Die im Rahmen der vorliegenden Arbeit implementierte Lösung deckt den Fall von Datenlücken ab. Fällt die Detektion des Schachbretts aufgrund einer Verdeckung des Roboters im Bild aus, werden die Odometrie-Daten so lange gesammelt, bis das Schachbrett wieder erkannt wurde. Damit ist es möglich, von der Kamera nicht sichtbare Bereiche per Roboter zu erkunden oder den Sichtbereich kurzzeitig zu verlassen. Kommen zwischen zwei Kamera-Posen weniger als zwei der Odometrie an, wird die aktuelle Kamera-Pose verworfen und die nächste abgewartet, um der Odometrie Zeit für zusätzliche Messungen zu lassen.

3.5.3 Kartierung

Zu der zuvor bereits vereinfachend getroffenen Annahme, Boden und Schachbrett seien parallel, lässt sich für die Kartierung eine weitere ergänzen: Hindernisse stehen auf dem Boden und haben bis zu einer Höhe, die die des Roboters übertrifft, eine Ausrichtung senkrecht nach oben. Es gäbe also keine Stufen, Schrägen oder Überhänge.

Ähnlich wie schon im Fall des Roboter-Rotationszentrums, lässt sich die Ausdehnung des Roboters in der Höhe damit ignorieren: Es wird angenommen, dass die zu kartierende Ebene der des Schachbretts entspricht. Also Ultraschall- sowie Tastsensor in dieser einen Ebene messen und nicht, wie tatsächlich, in unterschiedlicher Höhe montiert sind. Außerdem wird damit die vertikale kegelförmige Ausbreitung des Ultraschalls vernachlässigt.

Die zu erstellende Karte wird als Rasterkarte (Kapitel 2.2.2) gespeichert, die Auskunft darüber gibt, ob eine Rasterzelle belegt, frei oder unbestimmt ist. Die durch den Roboter erfassbaren Informationen stammen von dessen Stoßstange und seinem Ultraschallsensor. Hinzu kommt die Information, dass die Position, an der sich der Roboter befindet, frei sein muss.

Ausgehend von der Pose des Roboters werden diese Daten, wie in den vorherigen Unterkapiteln erläutert, in das übergeordnete Roboterkoordinatensystem transformiert. Dazu müssen sie auf diese Pose bezogen werden. Für die Markierung der aktuellen Roboterpose als „frei“, ist das direkt möglich. Für den Tastsensor muss die Lage der Stoßstange relativ zur Drehachse des Roboters in dem lokalen Roboterkoordinatensystem vorliegen. Sie ist mittig in einem festen Abstand vom Drehzentrum in Fahrtrichtung angebracht. Auch der Ultraschallsensor ist geradeaus parallel zu dem Schachbrett blickend hinter der Stoßstange und darüber montiert. Abbildung 3.14 zeigt dies in der Aufsicht inklusive einem in dem Sensor gelagerten Sensorkoordinatensystem und einem Hindernis, dessen Distanz als Kreisbogen gemessen wird.

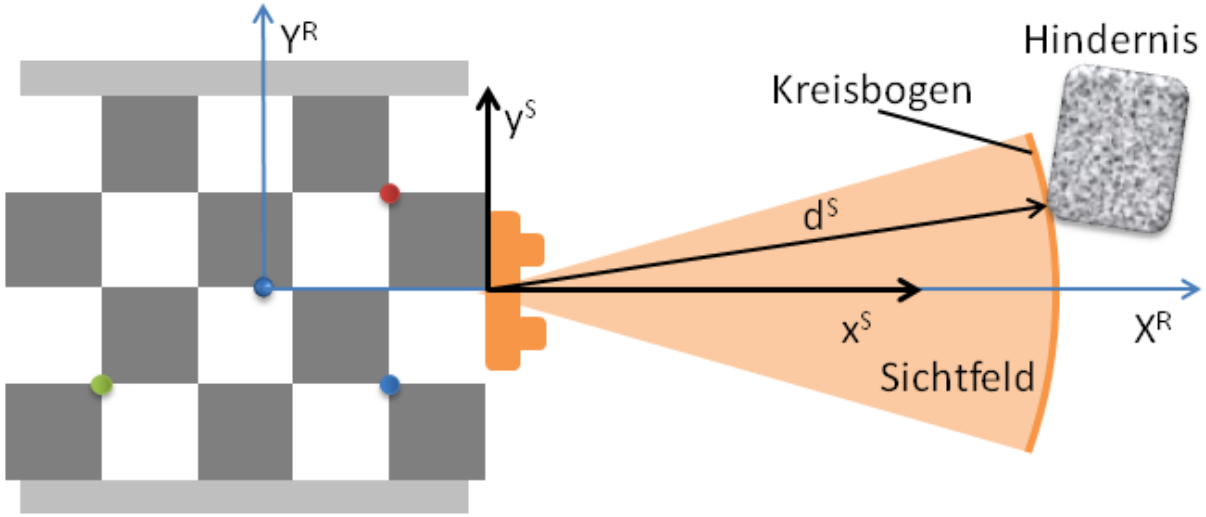

Abbildung 3.14: Roboter- und Sensorkoordinaten
Schematische Darstellung einer Aufsicht des Roboters mit dessen lokalem Koordinatensystem (X^R, Y^R) sowie dem im Ultraschallsensor gelagerten Sensorkoordinatensystem (x^S, y^S). Die in letzterem gemessene Distanz d^S zu einem Hindernis im Sichtfeld des kegelförmig ausgestrahlten Ultraschalls ist die Entfernung zu dem Kreisbogen – die genaue Richtung zu dem Hindernis ist unbekannt. Da die Messung in ein Raster abgebildet werden soll, ist deshalb ein 2D-System angesetzt, obwohl es sich bei einer Richtungsmessung um eine eindimensionale handelt.

Je nach Größe der Rasterzellen müssen die Ausmaße der Messungen der drei Kartierungsmöglichkeiten bei der Markierung der entsprechenden Rasterzellen beachtet werden. Sind die Rasterzellen deutlich kleiner als der Roboter, steht er auf mehreren Zellen, die um die aktuelle Pose herum als frei zu markieren sind. Die Stoßstange hat ebenfalls eine Ausdehnung und kann daher mehrere Zellen als belegt markieren. Hierbei ist jedoch nicht sicher, welche dieser Zellen dann tatsächlich belegt ist und dadurch den Taster auslöst. Für den Ultraschallsensor ist die kegelförmige Ausbreitung des Signals zu beachten: je weiter entfernt ein Objekt detektiert wird, desto größer wird der Kreisbogen, auf dem es liegen kann. Dieser Kreisbogen kann also auch mehrere Zellen betreffen. Außerdem ist davon auszugehen, dass alle Zellen zwischen dem Sensor und dem Kreisbogen frei sind. Ein dort liegendes Ziel wäre sonst detektiert worden.

Der vorausgehende Absatz schildert mehrere unsichere Annahmen. Genau genommen dürfte nur die Zelle als belegt markiert werden, in der der Teil der Stoßstange oder des Kreisbogens liegt, der tatsächlich Kontakt mit einem Ziel hat. Da nur ein Tastsensor eingesetzt wird, ist unbekannt, an welcher Stelle die Stoßstange ein Hindernis trifft. Gleiches gilt für den Ultraschallsensor: Es lässt sich aus einer Messung nicht rekonstruieren, wo auf dem Kreisbogen das Signal reflektiert wird.

Aufgrund dieser Unsicherheiten, wird ein Raster mit Wahrscheinlichkeiten verwendet. Die Pose trägt dazu bei, die Wahrscheinlichkeit der Zellen zu erhöhen, dass diese frei sind. Der Tastsensor tut dies für belegte Zellen und die Ultraschallmessung steuert beides bei. Da die Unsicherheiten dabei unterschiedlich groß sind, werden diese verschiedenen Messmethoden unterschiedlich gewichtet: Es wird angenommen, dass Zellen, die der Roboter durchquert, viel sicherer frei sind als solche, in denen per Ultraschall nichts detektiert wurde. Verschiedenste Szenarien, in denen hierbei Hindernisse aufgrund deren Größe oder Oberflächenbeschaffenheit „übersehen“ werden, finden sich in den Ergebnissen dieser Arbeit. Die Markierung einer Zelle als belegt ist per Ultraschall unsicherer als die mittels Tastsensor. Es wird angenommen, dass letzterer nahezu jedes Hindernis detektiert. Außerdem ist je weiter ein Ziel entfernt ist, die Anzahl der infrage kommenden Zellen größer: Der Kreisbogen ist länger als die Stoßstange. Es sind jeweils alle fraglichen Zellen gleichzeitig mit dem gleichen Wert zu aktualisieren.

Das Gebiet muss mit dieser Vorgehensweise mehrfach mit Messungen abgedeckt werden, um ausreichend hohe Wahrscheinlichkeiten für oder gegen eine Belegung der jeweiligen Zellen zu erhalten.

Abzustimmende Kenngrößen der Kartierung sind die folgenden:

- Rasterzellengröße
- Gewichtung zwischen Pose, Ultraschall und Tastsensor als Quelle
- Anzahl und Lage relevanter Rasterzellen für die drei Messmethoden
- Maximale und minimale Zielentfernung

Zu beachten sind Abhängigkeiten von den ebenfalls im Rahmen der Fahrstrategie relevanten Größen (siehe Kapitel 3.2). Die Beschreibung und Abstimmung dieser Größen erfolgt in Kapitel 4.

3.6 Kalibrierung

3.6.1 Kamerakalibrierung

Die Kamerakalibrierung als Prozess zur Ermittlung der inneren Orientierung der Kamera wird im Rahmen der vorliegenden Arbeit, wie aus der Nahbereichsphotogrammetrie bekannt, mit einem indirekten Verfahren durchgeführt. Die zu ermittelnden Parameter der Kamera sind die der in Kapitel 2.1.3.1 beschriebenen Kameramatrix und die zugehörigen Verzeichnungskoeffizienten. Sie werden Luhmann (2010) entsprechend per Ausgleichung mehrerer Beobachtungen eines ebenen Testfeldes also als Testfeldkalibrierung geschätzt. Die Kalibrierung erfolgt einmalig für die eingesetzte Kamera, da die innere Orientierung als konstant angenommen wird.

Das Testfeld bildet ein in DIN-A4 ausgedrucktes 9x6 großes Schachbrettmuster, dessen Ecken, wie in Kapitel 3.3, die Objektpunkte bilden. Damit stehen bereits in einem Bild mehr Beobachtungen als Unbekannte zur Verfügung. Da die Kalibrierung mit einem ebenen Testfeld jedoch mindestens zwei Aufnahmen aus unterschiedlichen Richtungen, Aufnahmewinkeln und zueinander gekantet erfordert, kommt eine deutliche Überbestimmung zustande. Für die Kalibrierung kann demnach ein kleineres Schachbrettmuster verwendet werden, wenn es um die Frage der mindestens erforderlichen Schachbrettpunkte geht, die benötigt werden, um die Freiheitsgrade dieser Aufgabe abzudecken. Um in einer Aufnahme eine möglichst gute Abdeckung des Bildes mit Beobachtungen zu erhalten, ist jedoch ein größeres Muster wie das hier verwendete vorzuziehen.

Um das gesamte Bildformat abzudecken, die Redundanz zu erhöhen und günstige Schnittbedingungen der Strahlen zu erhalten, werden jedoch deutlich mehr unterschiedliche Aufnahmen empfohlen. Luhmann (2010) nennt 16-25 Bilder für ein räumliches Testfeld. Diese werden hier erfasst, indem das Schachbrett entsprechend vor der Kamera positioniert und das Bild aus dem Videostrom entnommen wird.

Die Berechnung erfolgt in OpenCV mithilfe einer Funktion, die zunächst eine genäherte innere Orientierung berechnet, dann damit die äußere Orientierung in jedem Einzelbild schätzt und zuletzt mittels Ausgleichung die Fehlerquadratsumme minimiert, die sich durch die Projektion gemessener Bildpunkte anhand der geschätzten Orientierung zurück in Objektkoordinaten ergibt.

3.6.2 Bestimmung des Rotationszentrums

Für die Transformation der per Kamera beobachteten Schachbrettecken aus dem ebenen Schachbrettkoordinatensystem in das des Roboters ist, wie in Kapitel 3.4.3 gefordert, zunächst die Position des Roboterdrehzentrums relativ zu dem Schachbrett zu bestimmen. Dies geschieht einmalig oder

initial vor jedem Einsatz des Systems. Letzteres ist beispielsweise nötig, falls transportbedingt mit Verformungen des Roboters zu rechnen ist. In beiden Fällen ist das Prozedere identisch. Die Kamera beobachtet eine Umdrehung des Roboters um die eigene Achse. Die über diese Zeit gemessenen Schachbrettpunkte ergeben die lokalen Schachbrettkoordinatensysteme. Von diesen wird jeweils der Ursprung betrachtet. Damit liegt für jedes Bild ein Punkt vor. Diese Punkte beschreiben also den Kreis, auf dem sich der Ursprung des Schachbretts während der Drehung des Roboters bewegt hat. Dessen unbekannter Mittelpunkt entspricht dem Drehzentrum des Roboters. Seine Berechnung erfolgt mittels Kreisausgleichung.

Dabei wird von der allgemeinen Kreisgleichung

$$(x_i - x_m)^2 + (y_i - y_m)^2 = r^2 \tag{3.2}$$

mit Radius r, Mittelpunkt (x_m, y_m) und den Kreispunkten (x_i, y_i) ausgegangen. Diese ergibt ausmultipliziert und umgestellt Formel (3.3).

$$-2x_m x_i - 2y_m y_i + 1 \cdot (x_m^2 + y_m^2 - r^2) = -{x_i}^2 - {y_i}^2 \tag{3.3}$$

Diese lässt sich wiederum in die Matrixform

$$A \cdot x = l \tag{3.4}$$

bringen. Wobei A der Designmatrix, l dem Beobachtungsvektor und x den Parametern im Sinne einer Ausgleichung entsprechen. Hier werden jedoch nicht die Kreispunkte als Beobachtungen und die drei Parameter des Kreises (Mittelpunkt und Radius) als Parametervektor betrachtet. Dies ließe sich beispielsweise mittels Gauß-Helmert-Modell lösen (Niemeier, 2002), wobei die Beobachtungsgleichungen nicht linear sind. Die damit nötigen Näherungswerte und eine Iterative Ausgleichung können umgangen werden, indem direkt der algebraische Zusammenhang (3.3) minimiert wird. Hierzu werden die Parameter α aus (3.3) so gewählt, dass lineare Beobachtungsgleichungen entstehen.

$$\alpha_1 = -2x_m;\ \alpha_2 = -2y_m;\ \alpha_3 = x_m^2 + y_m^2 - r^2 \tag{3.5}$$

Die rechte Seite der Gleichung 3.2 bildet den Beobachtungsvektor l, die linke mit den eingesetzten Parametern α den funktionalen Zusammenhang (3.6), der zu Designmatrix und Parametervektor führt.

$$\mathrm{f_i}(\alpha_1, \alpha_2, \alpha_3) = l_i \tag{3.6}$$

Die Designmatrix enthält gemäß Niemeier (2002) als Koeffizienten a_{ij} die partiellen Ableitungen nach den Parametern.

$$a_{ij} = \left(\frac{\partial \mathrm{f_i}}{\partial \alpha_\mathrm{j}}\right) \tag{3.7}$$

Formel (3.4) ausgeschrieben lautet also

$$[x_i \quad y_i \quad 1] \cdot [\alpha_1 \quad \alpha_2 \quad \alpha_3] = [-{x_i}^2 - {y_i}^2] \tag{3.8}$$

Die beobachteten Kreispunkte werden als gleichgenau und unkorreliert angenommen, weshalb kein gesondertes stochastisches Modell angesetzt wird.

Die eigentliche Ausgleichung ist die Berechnung der geschätzten Parameter $\hat{x}$.

$$\hat{x} = (A^T \cdot A)^{-1} \cdot A^T \cdot l \tag{3.9}$$

Das Umstellen von (3.5) ergibt die gesuchten Koordinaten des Mittelpunkts und den Radius des ausgeglichenen Kreises.

$$\hat{x}_m = -\frac{\hat{\alpha}_1}{2};\ \hat{y}_m = -\frac{\hat{\alpha}_1}{2};\ \hat{r} = \sqrt{\hat{x}_m^2 + \hat{y}_m^2 - \hat{\alpha}_3} \tag{3.10}$$

Mit den Koordinaten des Mittelpunkts ist das Rotationszentrum des Roboters bezogen auf das Schachbrettkoordinatensystem damit bestimmt.

Angemerkt sei: Für eine Genauigkeitsbetrachtung oder Ausreißertests werden Residuen benötigt, die hier aus der Differenz zwischen der Distanz der beobachteten Kreispunkte (x_i, y_i) zum Kreismittelpunkt $(\hat{x}_m, \hat{y}_m)$ und dem Radius $\hat{r}$ berechnet werden können:

$$v_i = \sqrt{(x_i + \hat{x}_m)^2 + (y_i + \hat{y}_m)^2} - \hat{r} \tag{3.11}$$

Denn die Residuen direkt aus der Ausgleichung $v_i = A \cdot \hat{x} - l$ beziehen sich auf den Beobachtungsvektor des algebraischen Zusammenhangs und nicht auf die Kreispunkte.

3.6.3 Roboter

3.6.3.1 Kalibrierung des Antriebs

Die für die Odometrie benötigten Größen des Roboterantriebs, Radumfang und –abstand, werden durch eine Kalibrierung bestimmt. Der Umfang des Kettenantriebs bezieht sich nicht auf den Umfang der gesamten Kette, sondern auf den, eines die Kette führenden Rades inklusive der darüber laufenden Kette. Der Abstand ist bis zu der Mitte der Ketten zu messen, da diese dem Modell nach keine Ausdehnung haben. Zunächst ist der Radumfang zu ermitteln, da er unabhängig von dem Abstand ist, solange der Roboter geradeaus fährt. Fährt er eine Kurve oder dreht sich im Kreis, kommt erst der Abstand der Räder als benötigte Größe hinzu.

Es werden verschiedene Distanzen (30, 50, 100 und 150 cm) mit Klebeband auf ebenem Boden markiert. Diese fährt der Roboter jeweils von einer 0 cm-Marke startend mehrmals hintereinander. Der Versuch wird mit unterschiedlichen Geschwindigkeiten und mit unterschiedlichen Beschleunigungen durchgeführt. Für die Ermittlung der gesuchten Werte wird für den Prototyp jedoch nur eine Strecke von 50 cm bei einer festgelegten Geschwindigkeit genutzt (siehe Kapitel 4.1). Der zunächst als Näherungswert per Hand gemessene Radumfang wird so angepasst, dass die zurückgelegte Strecke des Roboters der programmierten entspricht.

Ist der Radumfang bekannt, wird der Roboter angewiesen eine und mehrere Drehungen um die eigene Achse durchzuführen. Auch dieser Versuch wird mit unterschiedlichen Geschwindigkeiten und Beschleunigungen durchgeführt. Die Gradzahl, um die die tatsächliche Drehung von der vorgegebenen abweicht, ermöglicht wiederum, den gemessenen Abstand der Ketten anzupassen.

3.6.3.2 Untersuchung der eingesetzten Sensoren

Frischknecht & Other (2006) haben sämtliche Sensoren des Mindstorms-Bausatzes in einer Vorabversion getestet. Für den Tastsensor werden keine eigenen Tests durchgeführt. Einige der beschriebenen Tests für den Ultraschallsensor werden wiederholt und erweitert, um zu überprüfen, ob die geschilderten Charakteristiken auch auf den hier eingesetzten zutreffen.

Es werden verschiedene Distanzen zu unterschiedlich geformten und beschaffenen Zielen in den beiden in Abschnitt 2.3.1.3 genannten Messmodi gemessen. Der Sensor befindet sich dabei in der gleichen Höhe, wie er am Roboter montiert ist, und misst parallel zum Boden. Getestet wird auch der Winkelbereich, in dem sich das Signal kegelförmig ausbreitet.

3.7 Implementierung

Eine Übersicht über die Pakete und Dateien der im Folgenden beschriebenen Programmierung findet sich im Anhang dieser Arbeit.

3.7.1 Roboter

Neben Programmen für die Kalibrierung des Roboters und zum Testen der Sensoren, werden zwei wichtige Bereiche für das vorliegende Projekt umgesetzt. Dies sind die im nachfolgenden Kapitel beschriebene Kommunikation per Bluetooth mit dem PC und die gesamte Umsetzung der Fahrstrategie aus Kapitel 3.2 inklusive der Odometrie. Die Programme werden am PC entwickelt und die fertig erstellten ausführbaren Programme per Bluetooth auf den NXT übertragen.

Wurde die Bluetoothverbindung durch das gestartete Programm auf dem NXT erfolgreich hergestellt, läuft das in Kapitel 3.6.2 vorgestellte Prozedere zur Ermittlung des Rotationszentrums ab. Diese Zeit wird außerdem für den Zeitabgleich zwischen PC- und NXT-Uhr genutzt. Anschließend beginnt der Roboter autonom entsprechend der Fahrstrategie seine Umgebung zu erkunden und überträgt seine Messungen fortwährend an den PC, dem dann der Offset der NXT-Uhr bereits bekannt ist. Die Ermittlung der Pose geschieht parallel (pseudoparallel, da der NXT nur einen Rechenkern besitzt) in einem gesonderten Prozess (englisch: Thread). Dieser ist so programmiert, dass er alle 250 Millisekunden die Pose des Roboters liefert , bei jedem Stoppen des Roboters die gemessene Distanz und den Zustand des Tastsensors abruft und für die Kartierung an den PC schickt.

3.7.2 Kommunikation mit dem Roboter

Die Kommunikation bildet ein eigenständiges Modul, dessen Klassen sowohl vom Roboter, als auch vom PC genutzt werden. Abgesehen von der Übertragung der Systemzeiten, bei der der PC in der Funktion als *Client* einen Zeitstempel an den als *Server* fungierenden Roboter sendet und daraufhin eine Antwort erhält, kommt das *publish-subscribe* (entspricht zu Deutsch: *veröffentlichen und abonnieren*) Entwurfsmuster zum Einsatz. Für die Übertragung der Sensordaten und der Roboterpose fungiert der Roboter als *Publisher*, der Daten eines bestimmten Typs veröffentlicht, die von dem als *Subscriber* auftretenden PC empfangen werden. Dieser muss zuvor lediglich angemeldet haben, dass er Interesse an diesen Daten hat. Diese Vorgehensweise hat den Vorteil, dass die Reihenfolge, in der Daten unterschiedlichen Typs übertragen werden, beliebig ist und asynchron erfolgt (Siciliano & Khatib, 2008). Ein Datentyp entspricht hierbei einem Protokoll, das festlegt, wie die Daten für die Übertragung über einen seriellen Datenstrom serialisiert (Kodierung) werden und wie sie diesem wieder entnommen werden müssen (Dekodierung). Die Protokolle sind die folgenden mit dem aufgeführten Inhalt:

- Kartierung (*MapProtocol*): Zeitstempel (Roboter-Zeit), Distanz, Tasterzustand
- Pose (*OrientationProtocol*): Zeitstempel (Roboter-Zeit), Pose (zwei Translationen, ein Winkel)
- Zeit (*TimeStampProtocol*): Sendezeitpunkt (PC-Zeit), Empfangszeitpunkt (Roboter-Zeit)

Jeweils auf dem PC und auf dem Roboter ist ein gesonderter Thread für die Kommunikation zuständig.

3.7.3 Verarbeitung der Daten des Roboters

Auf dem PC erfolgt die Berechnung der Karte. Diese wird, wie von Berns & Schmidt (2010) vorgeschlagen, in einer eigenen Klasse (*GridMap*) mithilfe einer *Hashmap* implementiert. Dabei handelt es sich im Vergleich zu einem statischen Array um eine dynamisch erweiterbare Datenstruktur, die Paare von Schlüsseln und Werten aufnimmt. Die Schlüssel bilden hier die zweidimensionalen Roboterkoordinaten (Klasse *GridPoint*), als Wert wird ein Objekt der Klasse *GridSquare* gespeichert, die eine Rasterzelle repräsentiert. In ihr steht also die Wahrscheinlichkeit der Zelle, frei oder belegt zu sein. Weitere Attribute sind leicht zu ergänzen.

Treffen nun Messungen des Roboters ein, wird zunächst dessen Pose zu diesem Zeitpunkt (korrigiert mithilfe der Kamera; siehe Abschnitte 3.5.2 und 3.7.4) herangezogen und von dieser ausgehend die Karte aktualisiert. Die entsprechenden Koordinaten der betroffenen Rasterzellen werden in der Hashmap als Schlüssel gesucht und, falls sie noch nicht existieren, in Form eines neuen Eintrags angelegt. Die Wahrscheinlichkeit wird eingetragen beziehungsweise aktualisiert. Für jede Zelle gibt es also nur einen Eintrag in der Hashmap, egal wie oft sie gemessen wird. Die endgültige Karte wird durch Auslesen der Hashmap und Eintragen der entsprechenden Werte in einen Rasterplot erstellt. Dabei gilt: Je höher der Wert, desto sicherer ist die Zelle belegt, je kleiner er ist, desto sicherer ist sie frei.

Das Aktualisieren der Karte geschieht in der GridMap-Klasse, indem die drei in Kapitel 3.5.3 beschriebenen Messmethoden unterschieden und entsprechend behandelt werden. Für eine Ultraschallmessung liegen beispielsweise dessen Sensorkoordinatensystem und die gemessene Distanz vor. Die Berechnung der als frei zu markierenden Zellen zwischen Kreisbogen und Sensor erfolgt mithilfe des Bresenham-Algorithmus (Kapitel 0). Sind die Rasterzellen groß genug und die Distanz entsprechend kurz, liegt der Kreisbogen innerhalb einer einigen Zelle. Somit ist es dabei vereinfachend möglich den Kreisbogen auf einen Punkt zu reduzieren, so dass nur eine Gerade in der Blickrichtung des Roboters zu rastern ist.

3.7.4 Verarbeitung der Daten der Kamera

Die Daten der Kamera, also deren Videostrom wird am PC unter Einsatz von OpenCV folgendermaßen ausgewertet. Es wird ein Bild bearbeitet und dann mit dem aktuellsten fortgefahren – Bilder, die noch während der Berechnung eintreffen, werden unter Umständen verworfen.

Der erste Schritt besteht in der Detektion des Schachbretts, die mithilfe einer entsprechenden OpenCV-Funktion (*findChessboardCorners*) stattfindet und in einem weiteren Schritt subpixel-genau verfeinert wird (*cornerSubPix*). Bei der Detektion des Schachbretts erfolgt zunächst eine Umrechnung des anhand des Histogramms normalisierten Bildes in schwarz und weiß mit einem adaptiven Schwellwert. Dies wird über Flags der Funktion mitgeteilt. Intern erfolgt die Suche dann mithilfe morphologischer Operatoren und einer Suche nach dem Schachbrettmuster der zuvor mitgeteilten Größe. Ist dieses teilweise verdeckt, wird es nicht gefunden. Außerdem muss ein ausreichend großer weißer Rand bleiben, damit die Segmentierung der äußeren schwarzen Kästchen gelingt. Die Verfeinerung der gefundenen Schachbrettecken erfolgt iterativ in einem Suchfenster, dessen Größe anzugeben ist. Außerdem werden Abbruchbedingungen für die Iteration übergeben: Sie wird beendet wenn entweder eine bestimmte Zahl von Iterationen erreicht ist oder die Veränderung der Position der Ecken zwischen zwei Iterationen einen Wert unterschreitet.

Anschließend werden die Koordinaten der Schachbrettecken anhand der kalibrierten inneren Orientierung der Kamera entzerrt. Dies spart gegenüber der Entzerrung des ganzen Bildes vor der Detektion Rechenzeit. Handelt es sich um das erste Bild nach dem Start des Programms, wird nun die Ho-

mographie für die Definition des Objektkoordinatensystems gemäß Abschnitt 3.4.2 geschätzt (*findHomography*). Die Funktion wird mit den Standardübergabeparametern genutzt: Es wird davon ausgegangen, dass es keine Ausreißer gibt – alle Punktkorrespondenzen fließen in die Ausgleichung nach kleinsten Quadraten ein.

Liegt die Homographie vor, wartet das Programm bis sich der Roboter für die Ermittlung des Drehzentrums zu drehen beginnt. Dies wird festgestellt, indem die Bewegung des Schachbretts zwischen den Bildern beobachtet wird. Es folgt die Schätzung des Drehzentrums gemäß Abschnitt 3.6.2, sobald der Roboter sich mindestens 350° gedreht hat und mindestens 18 Bilder erfolgreich detektiert wurden.

Bevor der Roboter seine Erkundung startet, hält er inne und setzt den Ursprung des Roboterkoordinatensystems auf seine aktuelle Pose. In dieser Zeit wird die Homographie erneut berechnet, so dass Objekt- und Roboterkoordinatensystem die Abschnitt 3.4.3 entsprechenden Beziehungen aufweisen.

Die nun folgende ständige Aktualisierung der Roboterpose und deren Umrechnung in Roboterkoordinaten unter Berücksichtigung des ermittelten Drehzentrums erfolgt noch in C++. Die Pose wird dann aber für die Aktualisierung der Karte in dem in Java geschriebenen Programm benötigt. Die dazu nötige Übertragung erfolgt dateibasiert: Jede Kamerapose wird inklusive Zeitstempel in eine Datei in ein Verzeichnis geschrieben. Dieses wird von dem Java-Programm überwacht, das die Pose erhält, indem es die Datei ausliest und anschließend löscht. Die gemäß Abschnitt 3.5.2 verwendete Warteschlange für die Odometrie-Posen fängt bei diesem Vorgang entstehende Verzögerungen auf.

4 Ergebnisse

4.1 Roboter

Die Planungs- und Konstruktionsphase des Roboters wurde bereits erläutert. Auch dabei aufgetretene Probleme, die zu Modifikationen des Prototyps führten. Im Folgenden geht es um Tests und die Kalibrierung einzelner Komponenten sowie die konkrete Wahl der für die Fahrstrategie relevanten Größen.

4.1.1 Fahrstrategie

In Kapitel 3.2 sind die für die Fahrstrategie relevanten Größen bereits aufgeführt. Deren Wahl wird im Folgenden erläutert. Die Argumente stützen sich dabei auf Erfahrungen, die durch das Testen des Roboters in der Testumgebung gewonnen wurden.

Fahrgeschwindigkeit und Drehgeschwindigkeit

Die Geschwindigkeit wird (abgesehen vom Beschleunigen und Bremsen) nicht variiert. Dies vereinfacht die Berechnungen und die Kalibrierung. Fahrgeschwindigkeit und Drehgeschwindigkeit zu variieren, bringt für das getestete Szenario keinen Mehrwert. Erst wenn der Roboter die Karte seiner Umgebung nutzen kann, wäre ein solcher denkbar, indem er weniger gut kartierte Bereiche schneller erreichen und sich für gründlichere Messungen dort langsamer bewegen könnte.

Zwei Hauptargumente sind für die Wahl der Geschwindigkeit aufzuführen: Je langsamer sich der Roboter bewegt, desto länger dauert die Erfassung seiner Umgebung. Je schneller er sich bewegt, desto höher werden die Anforderungen an die zeitliche Synchronisation und desto eher treten Probleme bei der Detektion des Schachbretts auf. Die gewählten Werte (10 cm/s und 10°/s) erweisen sich als geeignete Abwägung zwischen diesen beiden Argumenten.

minimal zugelassene Hindernisentfernung:

Bei dieser Größe ist abzuwägen: Ist die Distanz groß, muss sich der Roboter kaum bewegen, sondern erfasst die Umgebung von einem Standpunkt aus. Aufgrund des Ultraschallkegels ergeben sich mit zunehmender Entfernung jedoch immer mehr Zellen, in denen das detektierte Ziel liegen kann (vergleiche Kapitel 3.5.3). Ecken oder Gänge werden so „übersehen“ und die Unsicherheit, welche Zelle tatsächlich belegt ist, nimmt zu. Ist der Wert zu klein, wird der Roboter sehr viel fahren müssen, um die Umgebung zu erkunden und läuft schneller Gefahr sich in Ecken festzufahren. Vergleichsweise gute Ergebnisse liefern hier 25-30 cm.

maximaler und minimaler zugelassener Drehwinkel:

Hier spielt insbesondere das Verhalten des Roboters in Ecken eine Rolle. Im Verlauf der Tests in der Testumgebung zeigte sich, dass der Roboter sich in diesen besonders lange aufhielt, weshalb die in Kapitel 3.2 geschilderte Regel eingeführt wurde: Mehrmaliges hin und her Drehen in entgegengesetzter Richtung wird unterbunden. Der entsprechende Drehwinkel liegt gleichverteilt zufällig zwischen 60° und 120°, rechts oder links herum.

Entfernung, die im Fall einer Kollision zurückgesetzt wird:

Löst der Tastsensor aus, steht der Roboter direkt vor einem Hindernis und kann sich deshalb nicht drehen, ohne von diesem verschoben zu werden – ein Vorgang, der durch die Odometrie nicht erfassbar ist. Er muss vor einer Drehung also soweit zurücksetzen, dass eine Drehung möglich wird. Der nötige Wert ergibt sich aus der maximalen Ausdehnung des Roboters. Wird sie von dessen Drehachse aus gerechnet und der entsprechende Abstand der Stoßstange abgezogen, ist das Ergebnis die nötige Distanz für das Zurücksetzen. Für den umgesetzten Prototyp findet sich die maximale Ausdehnung an den beiden hinteren Ecken der Kalibriertafel. Setzt er um 10 cm zurück, kann er sich um 180° drehen, ohne mit dem durch den Tastsensor detektierten Objekt zu kollidieren.

Ein sehr hilfreicher Schritt wäre an dieser Stelle die Berücksichtigung bereits kartierter Hindernisse, die einem Fahrmanöver hinderlich sind. Kollisionen bei Drehungen, die keiner der Sensoren erkennt, sind nämlich mit dieser Strategie möglich und erfordern in seltenen Fällen manuelles Eingreifen.

4.1.2 Sensoren

Frischknecht & Other (2006) haben eine Vorabversion des Lego Mindstorms NXT Sets umfangreich getestet. Für den Tastsensor ermittelten sie in vertikaler Ausrichtung, dass dieser ab einem Druck von 34 Gramm bzw. mit einer Kraft von 0,34 Newton auslöst. Da der Roboter solange fährt, bis der Sensor auslöst, wird diese Kraft zweifelsohne erreicht, falls die Stoßstange auf ein unbewegliches Hindernis trifft. Auf den Ultraschallsensor und die Kombination aus Motoren, Rotationsgebern und Kettenantrieb gehen die folgenden Absätze ein.

Lego gibt für den Ultraschallsensor eine Messgenauigkeit von +/-3 cm an, aber keine Information, ob es sich dabei um eine Standardabweichung handelt. Die maximal messbare Distanz wird mit 255 cm angegeben (LEGO Group, 2006). Frischknecht & Other (2006) führen folgende Ergebnisse ihrer Untersuchungen des Ultraschallsensors auf:

- Der Nullpunkt des Ultraschallsensors befindet sich im Zentrum der Sensorhülle.
- Der kleinste messbare Wert beträgt 3 cm.
- Messwerte werden mathematisch korrekt gerundet.
- Bei Distanzen bis 20 cm sind 95% der Messergebnisse auf 2 cm genau.
- Darüber gibt es Distanzbereiche, die gehäuft kein Ergebnis liefern oder einen grob falschen Wert.
- Je härter und glatter ein Ziel, desto besser messbar ist es.
- Je runder und kleiner ein Ziel, desto schwieriger ist es zu detektieren – insbesondere mit zunehmender Distanz.
- Der Reflektionswinkel hat großen Einfluss: Ändert sich der Winkel, in dem der Sensor auf eine Objektoberfläche misst, nur gering, kann dies ausreichen, um die Detektion unmöglich zu machen.
- Ein horizontal ausgerichteter Sensor erzielt bessere Ergebnisse, als ein vertikaler: Dasselbe Objekt wird horizontal in 160 cm noch erkannt, vertikal bei über 80 cm Entfernung nicht.
- Insbesondere für kleine Distanzen < 20 cm werden Objekte erkannt, die bis zu 30° von der Hauptmessrichtung des Sensors entfernt sind. Darüber hinaus sind >7,5° häufig schon zu viel.
- Der Sichtbereich ist nicht symmetrisch: Bei horizontaler Ausrichtung, werden rechts vom Sensor liegende Objekte besser erkannt, was darauf zurückgeführt wird, dass der Sender rechts und der Empfänger links liegen.

Dynamische Sensorversuche durch Frischknecht & Other (2006), bei denen ein Roboter während einer Fahrt auf eine Wand zu und von dieser weg misst, ergaben kritische Entfernungsbereiche, in denen fälschlicher Weise der Wert 255 cm ausgegeben wurde. Zwischen 20 und 50 cm erhielten sie außerdem mehrfach die falsche Distanz 48 cm. Der Sensor gibt 255 cm aus, wenn kein Ziel detektiert wurde, weshalb ein kurzzeitiges Aussetzen der Messfähigkeit vermutet wird. Allerdings wurden die Versuche wiederholt und die falschen Werte zeigten sich gehäuft in denselben Distanzbereichen.

Die Konstruktion des Roboters berücksichtigt diese Eigenschaften des Ultraschallsensors bei dessen Anbringung und Ausrichtung. Außerdem sind lediglich Distanzmessungen bis zu 30 cm für die Kartierung relevant, was den Einfluss zufälliger grober Fehler in größeren Entfernungen minimiert.

Eigene Tests, bei denen ein Schuhkarton als Ziel diente, der wahlweise mit einem Handtuch umwickelt wurde, bestätigen, dass die Oberflächenbeschaffenheit den genannten großen Einfluss hat. Die Messung fällt mit dem Handtuch eher aus. Auch die mit der Distanz steigende Abhängigkeit von dem Auftreffwinkel des Signals auf die Objektoberfläche bestätigten eigene Versuche: Je weiter das Objekt vom Sensor entfernt ist, desto kleiner ist der Winkelbereich, den es zum Sensor einnehmen kann, um noch „gesehen“ zu werden. Der Schuhkarton wird bei Distanzen um 150 cm nur noch erkannt, wenn er orthogonal zum Sensor ausgerichtet ist. Eine Verdrehung um wenige Grad ergibt bereits keine Messung mehr. Distanzen bis zur maximalen Messdistanz von 255 cm sind nur nahezu orthogonal auf eine ganze Wand bzw. kaum möglich. Der Schuhkarton ist dafür bei weitem zu klein.

In dem im Rahmen der vorliegenden Arbeit, zusätzlich zu den Ausführungen von Frischknecht & Other (2006), getesteten *Ping*-Messmodus, der bis zu 8 Distanzen liefern soll, zeigen sich außer den tatsächlich erwarteten und vorhandenen Distanzen Mehrfachechos – und zwar umso häufiger, je glatter die Objektoberfläche ist. Liegen mehrere unterschiedlich weit entfernte Objekte versetzt zueinander im Sichtfeld des Sensors, ergeben sie in diesem Modus unterschiedliche zum Teil grob falsche Distanzen, je nachdem, ob einzelne Objekte entfernt oder hinzugefügt werden. Nebenechos, die keinem Ziel zuzuordnen sind, häufen sich mit mehreren Zielen. Die eigentlich gesuchten Distanzen werden durch Ausreißer verfälscht. Die Ergebnisse der Mehrfachechos hängen sogar davon ab, was auf der Rückseite des Sensors an reflektierenden Flächen liegt. „Guckt“ der Sensor durch ein Loch einer Holzplatte, führt diese zu Mehrfachechos. Dies wäre bei einem Einsatz des Ping-Modus also im Entwurf des Roboters zu berücksichtigen. Auf dessen Einsatz wurde jedoch aufgrund der unkalkulierbaren Mehrfachechos verzichtet. Für das vorliegende Projekt ist der kontinuierliche Modus besser geeignet.

4.1.3 Antrieb

Frischknecht & Other (2006) ermitteln für den Motor eine minimal mögliche Umdrehung von einem Grad, was dem Bereich entspricht, den die Firma Lego als Genauigkeit der Rotationsgeber aufführt. Sie geben außerdem an, der Motor benötige empirisch ermittelt 0,44 Sekunden für eine komplette Umdrehung bei maximaler Kraft. Allerdings beeinflusst die Anzahl gleichzeitig laufender Motoren dies, so dass dieser Wert nur für einen einzelnen Motor gilt: Werden zwei Motoren gleichzeitig angesteuert, verringert sich die Umdrehungsgeschwindigkeit.

Für die vorliegende Arbeit wurde ergänzend getestet, wie präzise eine vorgegebene Strecke mit dem gebauten Prototyp gefahren werden kann. Die Protokolle der Tests verschiedener Distanzen und unterschiedlicher Drehungen auf der Stelle mit unterschiedlichen Geschwindigkeiten und Beschleunigungen finden sich auf der Daten-CD (Kalibrierung>Roboter). Sie ergeben hohe Wiederholgenauigkeiten für einzelne Distanzen (nie mehr als 1 cm Unterschied zwischen den Wiederholungen auch bis

150 cm). Bei niedrigerer Beschleunigung ergeben sich Abweichungen der gefahrenen Strecke bei Maximalgeschwindigkeit von über 5 cm auf Holzuntergrund. Dies ist darauf zurück zu führen, dass der angegebene Beschleunigungswert auch auf das Ausrollen der Motoren beim Anhalten Einfluss hat. Je höher die Geschwindigkeit, desto mehr weicht die gefahrene Strecke von einer Gerade ab – und zwar vorwärts mehr als wenn der Roboter rückwärts fährt. Mögliche Ursachen sind asymmetrische Gewichtsverhältnisse bzw. Verspannungen innerhalb des als fest angenommenen Robotergerüsts. Da es nahezu symmetrisch gebaut ist, kommt eher letzteres in Frage. Hier sind jedoch weitere Untersuchungen wünschenswert.

Für die gewählte Fahrstrategie vereinfacht sich die Berücksichtigung einiger Effekte: Da der Roboter nur geradeaus fährt oder sich auf der Stelle dreht, rotieren die Räder jeweils gleichschnell (bei der Drehung des Roboters gegenläufig). Es werden, wie zuvor erläutert, nur eine Drehgeschwindigkeit und eine für die Geradeausfahrt verwendet. Die oben erwähnten Abhängigkeiten der beiden Motoren voneinander hinsichtlich der Umdrehungsgeschwindigkeit sind also konstant und müssen nicht berücksichtigt werden; der Roboter erreicht lediglich eine niedrigere Maximalgeschwindigkeit. Die entsprechend Abschnitt 3.6.3.1 ermittelten Größen Radumfang und Radabstand werden vereinfachend für diese festgelegte Geschwindigkeit auf eine Distanz von 50 cm kalibriert. Dies entspricht in der Testumgebung einer mittleren Entfernung. Für einen Teppich und eine glatte Holzoberfläche unterscheiden sich die kalibrierten Werte im Zentimeterbereich, was zeigt, dass hier neben den tatsächlichen Maßen, andere Effekte, wie das Rutschen der Ketten einen Einfluss haben. So ist es nicht verwunderlich, dass sich der Radabstand um mehrere Zentimeter von dem per Hand gemessenen Wert unterscheidet.

4.1.4 Zeiten und deren Synchronisation

Mit der gewählten Strategie der Zeitsynchronisation, insbesondere mit der nach jedem Neustart von 0 Millisekunden zählenden NXT-Uhr, geht eine Untersuchung der Zeiten einher. Verzögerungen bei der Datenübertragung werden durch die Verwendung von Warteschleifen aufgefangen. Dass der Zeitabgleich zwischen PC und NXT per Bluetooth den dabei auftretenden verzögerungsbedingten Fehler toleriert, wurde bereits erläutert. Doch es bleibt die Frage zu beantworten, wie gut die NXT-Uhr eigentlich arbeitet. Dazu wurden Zeitstempel über 10 Minuten alle 500 Millisekunden ausgetauscht. Der Drift der NXT-Uhr beträgt in einer solchen Zeitspanne um die 10 Millisekunden, was für die umgesetzte Strategie tolerierbar ist. Während die Zeitstempel des PC dabei genau 500 ms auseinander liegen, variiert ihr Abstand bei den zurückgesendeten Zeitstempeln des NXT um diesen Wert mit einer Standardabweichung von 16 ms (Tabelle siehe Daten-CD: Genauigkeitsuntersuchungen>Zeitabgleich NXT-PC). Diese Variation wird auf zufällige Änderungen der Verzögerungen bei der Übertragung und der nötigen Rechnungen zurückgeführt und macht deutlich, dass der regelmäßige Abgleich funktioniert.

Alles in allem hat sich die Zeit im Verlauf der Tests nicht als Problem dargestellt, da andere Fehlerquellen, insbesondere im Bereich des Roboterantriebs, größeren Einfluss haben. Die gewählten Intervalle für Zeitabgleich, Aktualisieren der Pose für die Sensormessungen, usw. haben sich als ausreichend für die (zum Teil geringen) Genauigkeiten und im Zusammenhang mit den, nicht zu hoch angesetzten, Geschwindigkeiten des Roboters erwiesen. Tritt beispielsweise eine Verzögerung von 100 ms auf, bewegt sich der Roboter in dieser Zeit maximal 1 cm.

4.2 Kamera

Dieses Unterkapitel führt relevante Ergebnisse im Zusammenhang mit der Kamera und der Bildmessung des Schachbretts auf.

4.2.1 Untersuchung der Signalisierung

Eine Untersuchung der Schachbrettdetektion ergibt, dass alle Punkte des Schachbretts gleichgenau detektiert werden. Da keine Ausgleichung durchgeführt wird, verwundert es auch nicht, dass Randpunkte sich hinsichtlich der Genauigkeit nicht von den anderen unterscheiden – jeder Punkt wird isoliert betrachtet. Die wiederholte Detektion eines ruhenden Schachbretts in der Testumgebung ergibt für die Punkte eine mittlere Standardabweichung von 0,03 Pixel. Eine Quaderkante verläuft dabei in der Länge durch etwa 30 Pixel. Dichter vor der Kamera, bei etwa 1,5 m, sind sogar 0,02 Pixel möglich. Für wiederholte Messungen eines auf dem Boden des Testgebiets liegenden unbewegten 4x3-Schachbretts ergibt sich eine mittlere Standardabweichung von 0,3 mm. Probleme ergeben sich jedoch, wenn die Kameraparameter (Belichtung, Schärfe, Filter, etc.) nicht optimal eingestellt werden. Die Bildkompression beziehungsweise die Bildverarbeitung der Kamera können an Kanten dann dazu führen, dass der Schachbrettpunkt zwischen zwei Positionen springt, weil sich zwei Schachbrettquader an ihren Ecken nicht direkt berühren. Die Kompression führt dazu dass eine Kante längs über mehrere Pixel „verschmiert“ wird. Die Subpixelschätzung muss daraus die genaue Kantenposition rekonstruieren.

Die Tests verschiedener Konfigurationen finden sich auf der Daten-CD (Genauigkeitsuntersuchungen>Bildmessgenauigkeit). Ein Einfluss des Winkels, um den das orthogonal zur Kamera ausgerichtete Schachbrett gedreht wird, konnte nicht festgestellt werden. Wird es zu dieser verkippt, ist eine Detektion bis weit über 45° Schrägansicht möglich. Die Genauigkeit nimmt dabei in einer Koordinatenrichtung ab. Je nach Beleuchtung scheitert die Detektion des Schachbretts ab einer Entfernung, in der ein Schachbrettquader der Länge nach in weniger als 5-10 Pixeln abgebildet ist. Da alle Punkte erkannt werden müssen, funktioniert die Erkennung bei einem kleineren Schachbrett vergleichsweise länger. Ein 4x3 Muster funktioniert bis etwa 5 Pixel, ein 9x6 nur bis etwa 10.

Um eine Aussage darüber treffen zu können, wie genau eine Strecke im Testgebiet per Kamera zu messen ist, wird eine Distanz von 100cm per auf dem Boden ruhendem Schachbrett gemessen (Sollstrecke per Maßband; Abbildung 4.1). Es kommt dieselbe Vorgehensweise der Transformation über eine Homographie zum Einsatz, wie bei der Kartierung. 32-faches Messen der Strecke mit unterschiedlichen Ausrichtungen des Musters ergibt eine mittlere Strecke von 98,68 cm bei einer Standardabweichung von 3,33 cm (siehe Daten-CD: Genauigkeitsuntersuchungen>Distanzvergleich). Dies legt nahe, dass der Maßstab für das Schachbrett nicht ganz exakt ermittelt wurde. Dies zeigt auch, dass der wesentlich kleinere Maßstab des Schachbrettmusters für die Ausdehnung des Testgebiets sehr kurz ist.

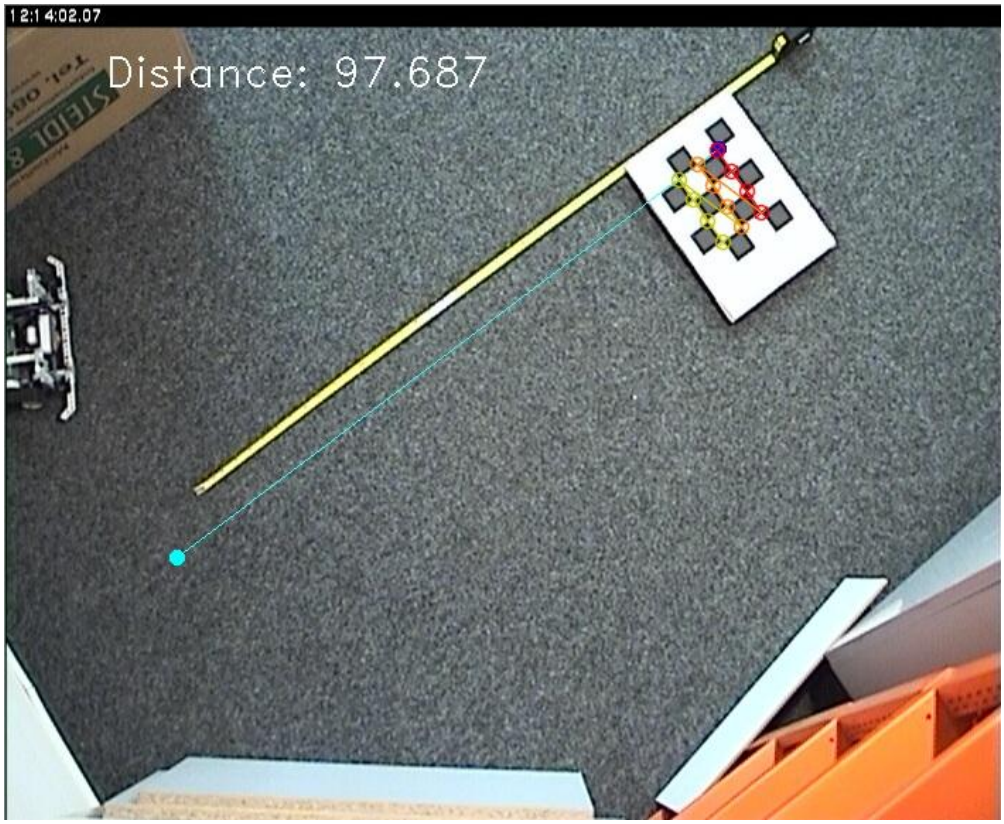

Abbildung 4.1: Versuchsaufbau: Distanzmessung per Schachbrett
Bandmaß in gelb. Startpunkt und Linie der Distanzmessung (bezogen auf den Ursprung des Schachbrettsystems) violett.

4.2.2 Drehzentrum

Die Kalibrierung des Drehzentrums wird mehrfach wiederholt, wobei der Abstand von der Kamera, die Antriebsart des Roboters, die Bodenbeschaffenheit und die Neigung des Bodens aus der Horizontalen heraus variiert werden. Da sie eine wesentliche Rolle bei der Verknüpfung des an der Kamera orientierten Objektkoordinatensystems mit dem des Roboters spielt, ist die Kenntnis der dabei erreichbaren Genauigkeiten für die Abschätzung der Genauigkeit der Roboterpose eine wichtige Einflussgröße. Die Testergebnisse (siehe Daten-CD: Kalibrierung>Drehzentrum) werden im Folgenden zusammengefasst.

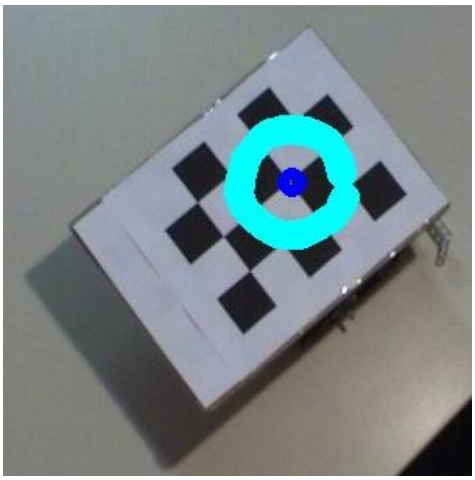

Abbildung 4.2: Kalibrierung des Drehzentrums mit Ketten

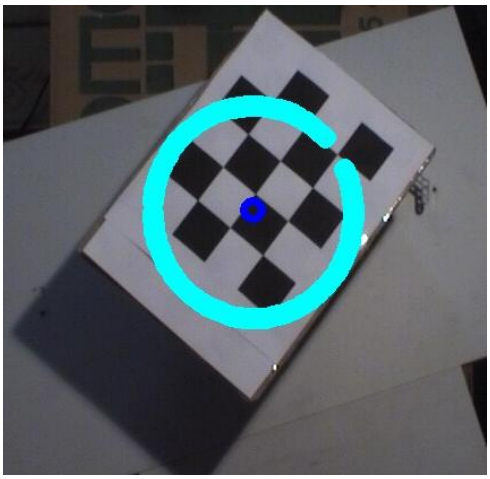

Abbildung 4.3: Drehzentrum mit Rädern

Mit dem Kettenantrieb variiert das ermittelte Rotationszentrum sowohl auf Teppich als auch auf glattem Holz im Bereich eines halben Zentimeters und ist damit deutlich ungenauer als die Bildmessungen des Schachbretts. Ursache sind die Fehler, die auch die Odometrie so ungenau machen, allen voran das unkontrollierbare Rutschen der Ketten, das bei der Drehbewegung besonders ausgeprägt ist. Aus diesem Grund wurden testweise die Ketten des Roboters durch Gummiräder ersetzt. Abbildung 4.2 und Abbildung 4.3 zeigen zum Vergleich die Trajektorien beider Antriebe, die exemplarisch für die Ermittlung des Drehzentrums sind. Die Position des Drehzentrums liegt mit dem geänderten Antrieb an einer anderen Stelle, da der Roboter einen entsprechend geänderten Radius fährt. Die Wiederholung des Tests zeigt eine wesentlich geringere Variation des ermittelten Zentrums mit den Rädern. Die Bestimmung des Rotationszentrums geschieht hier im Genauigkeitsbereich weniger Millimeter. Hier sind also die antriebsbedingten Fehler geringer. Der Antrieb kann jedoch in der verwendeten Form nicht auf Teppich eingesetzt werden, da die beiden Räder des Differentialantriebs dazu führen, dass der Roboter, wie in Abschnitt 2.2.4 erläutert, ein drittes Stützrad benötigt. Dieses blockiert bei Drehungen auf dem Teppich und verhakt sich mit diesem, was wiederum zu zufälligen Fehlern führt. Da alternative Antriebe im Rahmen der Arbeit nicht weiter umgesetzt und getestet werden konnten, bleibt festzuhalten, dass eine Veränderung des Antriebs die Bestimmung des Drehzentrums optimieren kann. Angesichts des großen Fehlers gegenüber dem Einfluss der Bildmessfehler, erscheint dies auch nötig und sinnvoll.

Die Unterschiede, abhängig von der Entfernung zwischen Kamera und Roboter während der Kalibrierung, fallen gering aus, da hierbei oben genannte Fehler unverändert bleiben und lediglich die Bildmessung davon profitiert, dass das Schachbrettmuster besser aufgelöst wird, je dichter es vor der Kamera ist.

Ein unerwartet großer Effekt zeigt sich, wenn die Bodenebene geneigt ist. Abbildung 4.4 zeigt, was passiert: Der Roboter rutscht schon bei geringer Neigung (ca. 5 Grad) je nach Ausrichtung während der Kreisfahrt, was die Kreisberechnung stark beeinflusst. Für die Kalibrierung des Drehzentrums muss also auf einen horizontalen Untergrund geachtet werden, sofern man die auftretenden Effekte nicht näher untersucht und, falls möglich, modelliert hat.

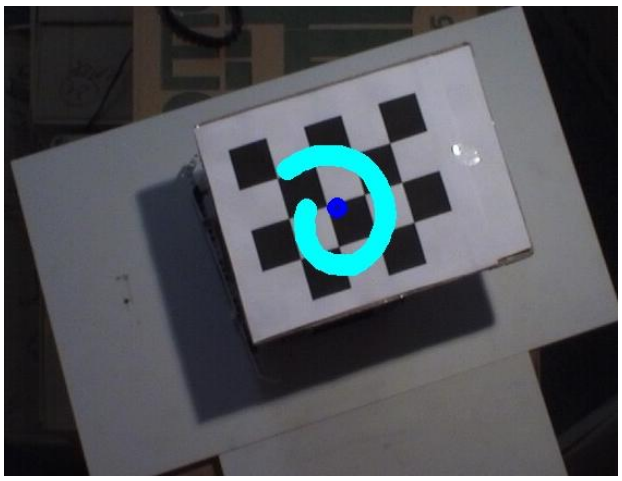

Abbildung 4.4: Drehzentrum mit geneigtem Untergrund

Neben dem Zentrum spielt die Ausrichtung des Schachbretts eine Rolle. Diese Abweichung der Nullrichtung des Schachbretts, von der des Roboters lässt sich überprüfen, indem dieser von der Startposition geradeaus fährt, was von der Kamera gemessen wird. Die unten folgende Abbildung 4.7 zeigt diese erste gefahrene Gerade. Die resultierende Abweichung in der Richtung ist mit unter einem

Zentimeter im Bereich der Odometriegenauigkeit. Eine gesonderte Kalibrierung der Richtung wurde nicht durchgeführt, sollte aber für höhere Genauigkeiten getätigt werden.

4.3 Kartierung

Der folgende Abschnitt geht zunächst auf die Wahl der für die Kartierung gesuchten Größen ein und betrachtet dann Trajektorien des Roboters sowie die resultierende Karte.

Rasterzellengröße, Anzahl und Lage relevanter Rasterzellen:

Bei der Größe der Rasterzellen steht im Allgemeinen die Verfügbarkeit des entsprechenden Speichers limitierend im Vordergrund. Da hier die Speicherung und Berechnung am PC erfolgt, ist dies kein Problem. Nichtsdestotrotz liefert die Implementierung in Form von Schlüssel-Objekt-Kombinationen bereits eine geeignete Struktur, um Speicherplatz effizient zu nutzen. Die Rasterzellengröße wird mit 10x10 cm relativ grob gewählt, erscheint jedoch ausreichend für die Testumgebung. Fehler, die dadurch entstehen, dass sich Tast- und Ultraschallmessungen vereinfachend jeweils nur auf die geradeaus liegenden Zellen beziehen, sind dadurch gering. Der Kreisbogen des Ultraschallmessers wird demnach nicht berücksichtigt und der Roboter selbst markiert auch nur die Zelle, in der sich sein Drehzentrum befindet, als frei. Für feiner aufgelöste Raster sollte dies entsprechend implementiert werden.

Maximale und minimale Zielentfernung:

Die maximale Zielentfernung, bei der ein Objekt in die Karte eingetragen wird, liegt bedingt durch die Fahrstrategie bei der zuvor beschriebenen Distanz, bei der der Roboter wegen eines Hindernisses anhält. Denn zu diesem Zeitpunkt wird dessen Position in der Karte gespeichert. Die minimale Entfernung liegt aufgrund der Beschränkungen des Ultraschallsensors bei 3 cm vor diesem. Bei geringen Entfernungen können sich gerade bei dem Tastsensor Messungen widersprechen, wenn die Rasterauflösung entsprechend gering ist: Die Messung könnte dieselbe Rasterzelle als belegt markieren, die die Roboterpose als frei annimmt.

Gewichtung zwischen Pose, Ultraschall und Tastsensor als Quelle:

Wie die Ergebnisse der Kartierung im Folgenden zeigen, gibt es hier viel Spielraum. Die Gewichtung kann für verschiedene Szenarien optimiert werden. Für den Testaufbau fließt die Pose als sicherste Beobachtung trotzdem nur mit dem Faktor +1 ein, da sie mehrfach markiert wird. Der Tastsensor markiert die Zelle, in der er ein Hindernis trifft, mit +2 und der Tastsensor als unsicherster markiert mit +1 die belegte Zelle und die dazwischen als frei angenommenen mit -1. Diese Werte sowie die Strategie, wann welche Messung zu einem Aktualisieren der Karte führt, sind jedoch weiter zu untersuchen und hier nicht abschließend geklärt.

In den vorausgehenden Abschnitten wurden bereits verschiedene Fehlerquellen gefunden, die sich auf die Karte der Umgebung auswirken. Die getroffene Annahme, Schachbrettebene und Boden seien zu jeder Zeit parallel, stellt ebenfalls eine Fehlerquelle dar, die sich auf die Kartierung auswirkt. Dieser Fehler lässt sich abschätzen, wenn man die initial berechnete Homographie betrachtet. Ist deren zugrunde gelegte Ebene zu dem Boden verkippt, führt dies zu einem mit der Entfernung wachsenden Fehler. Alle folgenden Ebenen werden so transformiert, als würden sie in dieser Ebene liegen, sind jedoch wiederum zum Boden parallel oder jeweils für sich verkippt. Das führt dazu, dass ihre Punkte nicht in der Ausgangsebene der Homographie liegen und somit entsprechend Abbildung 4.5 in ihrer Position versetzt abgebildet werden. Die gezeigte Situation entspricht der nahezu von oben

gefilmten Szene dieser Arbeit. Der resultierende Fehler für einen Punkt im Bild entspricht dem von Luhmann (2010) für den Fall beschriebenen, dass ein Punkt bei einer Projektivtransformation nicht in der Ebene liegt (4.1).

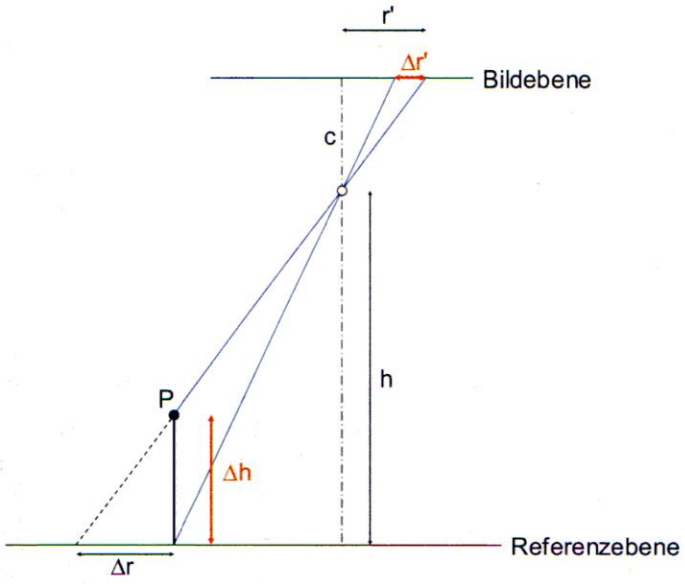

Abbildung 4.5: Lageversatz durch Höhenunterschied
Quelle: (Luhmann, 2010)

$$\Delta r = \frac{r'}{c}\Delta h \tag{4.1}$$

Der Lagefehler im Objektraum folgt laut Luhmann (2010) aus (4.2).

$$\Delta r = \frac{h}{c}\Delta r' = m_b \Delta r' \tag{4.2}$$

Umgestellt lässt sich damit berechnen, wie viel ein Punkt aus der Ebene abweichen darf, bevor er in der Lage um 1 cm abweicht (Δr). Folgende Daten repräsentieren die hiesige Situation: Eine Brennweite von $c = 8{,}4\ mm$, Zielentfernung von $h = 2{,}5\ m$ und ein maximaler Abstand von der Bildmitte $r'_{max} = 3{,}5\ mm$. (4.1) und (4.2) ergeben damit: $\Delta h = \frac{c}{r'_{max}}\Delta r = \frac{8{,}4\ mm}{3{,}5\ mm} \cdot 10\ mm = 24\ mm$

Die Anforderungen an ein exakt glattes Kalibrierungsmuster sind für die geringen Genauigkeiten dieses Aufbaus also gering. Wird jedoch die Homographie mit einem Schachbrett in einer Ecke des Bildes berechnet, das um 3° verkippt ist, kann dies auf einer Distanz von 2 Metern schlimmstenfalls zu einem Höhenversatz zwischen Signalisierungsebene und Boden von 10 cm führen. Wird dort ein Schachbrett detektiert und mit der Homographie transformiert, unterliegen dessen Punkte obigem Fehler und überschreiten die 2,4 cm deutlich.

Dies zeigt, dass der gewählte Weg über eine Homographie aus einer einzigen Schachbrettpose nicht optimal ist. Für höhere Genauigkeitsanforderungen ist ein komplexeres Verfahren nötig.

Abbildung 4.6: Versuchsansicht: Vergleich einer Strecke von 100 cm
Gezeigt ist der Roboter, der gerade die Strecke zurücklegt. Wann er sie erreicht hat, gibt die Odometrie an. Die Kamera misst ihrerseits unabhängig davon die zurückgelegte Distanz.

Um die Kameramessung weiter zu untersuchen, wird der Roboter im Kamerabild verfolgt während er, per Odometrie gemessen, eine Strecke von 100 cm zurück legt (Abbildung 4.6). Die Kameramessung nach 19 Wiederholungen ergibt im Mittel einer Strecke von 96,78 cm mit einer Standardabweichung von 3,29 cm (siehe Daten-CD: Genauigkeitsuntersuchungen>Distanzvergleich). Die im Mittel zu kurze Strecke ist mit einer ungenauen Kalibrierung sowohl der Roboterdaten (Radumfang und- abstand), als auch des Schachbrettmaßstabs zu begründen. Letzteres ist wahrscheinlich, da die Versuche in Abschnitt 4.2.1 dies bereits nahelegen.

Eine Beurteilung des gesamten Verfahrens lässt sich anhand der Trajektorien des Roboters tätigen. Abbildung 4.7 zeigt diese aus Odometrie bzw. Bildmessungen im Vergleich. Der Drift der Odometrie ist deutlich als vom Startpunkt größer werdende Differenz erkennbar. Insbesondere in Kurven addiert sich ein großer Fehler, der auch die Grenzen der Kalibrierung des Radabstands für den Roboter aufzeigt: Der Richtungsfehler geht hier immer in dieselbe Richtung, was auf eine nicht ausreichend genaue Kalibrierung hindeuten kann. Fehler in der Kalibrierung des Drehzentrums spiegeln sich wiederum dadurch wieder, dass dort, wo sich der Roboter auf der Stelle gedreht hat, die blaue Trajektorie anstatt eines Punktes einen Kreisbogen beschreibt – das falsche Drehzentrum dreht sich um das richtige.

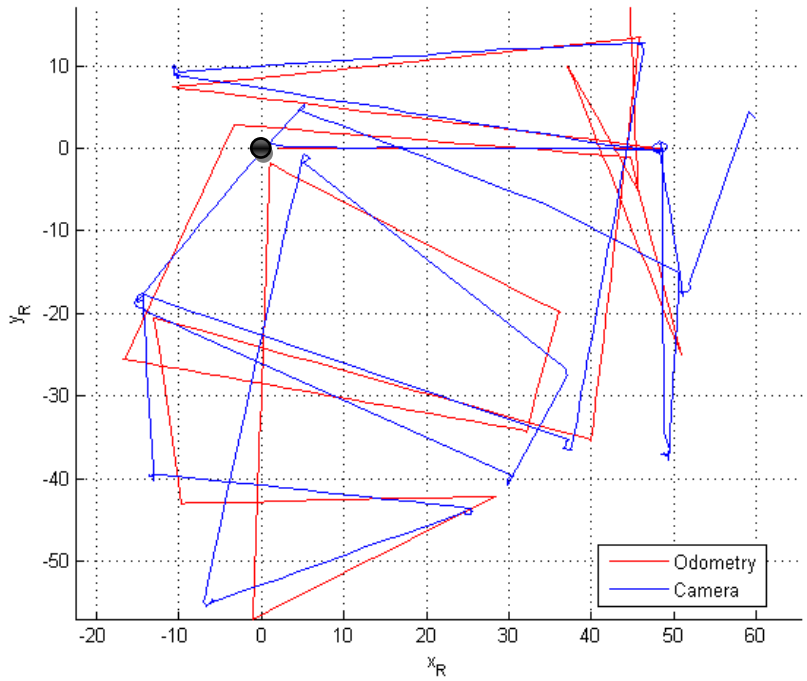

Abbildung 4.7: Trajektorien aus Odometrie bzw. Kamera
Beide Trajektorien sind auf das Drehzentrum des Roboters bezogen dargestellt. Der Startpunkt (Epoche 0) ist mit einem schwarzen Punkt markiert. Die Achsen sind die des übergeordneten Roboterkoordinatensystems in der Einheit cm.

Abbildung 4.8 zeigt die Trajektorie aus Bildmessungen über einen Zeitraum von 5 Minuten des Erkundens mit der beschriebenen Fahrstrategie. Es zeigt sich, dass die Effizienz dieser Strategie begrenzt ist: Der Roboter verweilt lange in Ecken, da er sich hier per Zufall dreht und diese entsprechend „zufällig“ wieder verlässt. Bestimmte Bereiche scheinen mehr, andere weniger häufig erreicht worden zu sein. Diese variieren jedoch auch mit jedem erneuten Versuch. Das macht deutlich, dass eine entsprechend lange Zeit angesetzt werden muss, um einigermaßen sicher das Gebiet abzudecken. Ein Versuch, das Testgelände um ein Hindernis im Zentrum zu erweitern, zeigt: Gibt es zu viele Ecken oder zu enge Bereiche in der Umgebung, werden diese selten oder überhaupt nicht erfasst, da der Roboter sie mit dieser Strategie nicht ausreichend umfahren und kartieren kann. Bei engen Gängen, in die der Roboter mit richtiger Ausrichtung hineinpassen würde, ist die Wahrscheinlichkeit, dass der Ultraschallkegel eine der Seiten erreicht und der Roboter deshalb umkehrt, so hoch, dass er nicht hinein fährt. Dahinter liegende Bereiche finden sich deshalb, genau wie der Gang selber, nicht in einer resultierenden Karte wieder.

Abbildung 4.8: Trajektorie aus Kamerasicht über 8 Minuten
Jeder Kringel repräsentiert den Ursprung des Roboters (nicht dessen Drehzentrum!) in einem Bild. Diese wurden über die Zeit in das aktuelle Kamerabild geplottet.

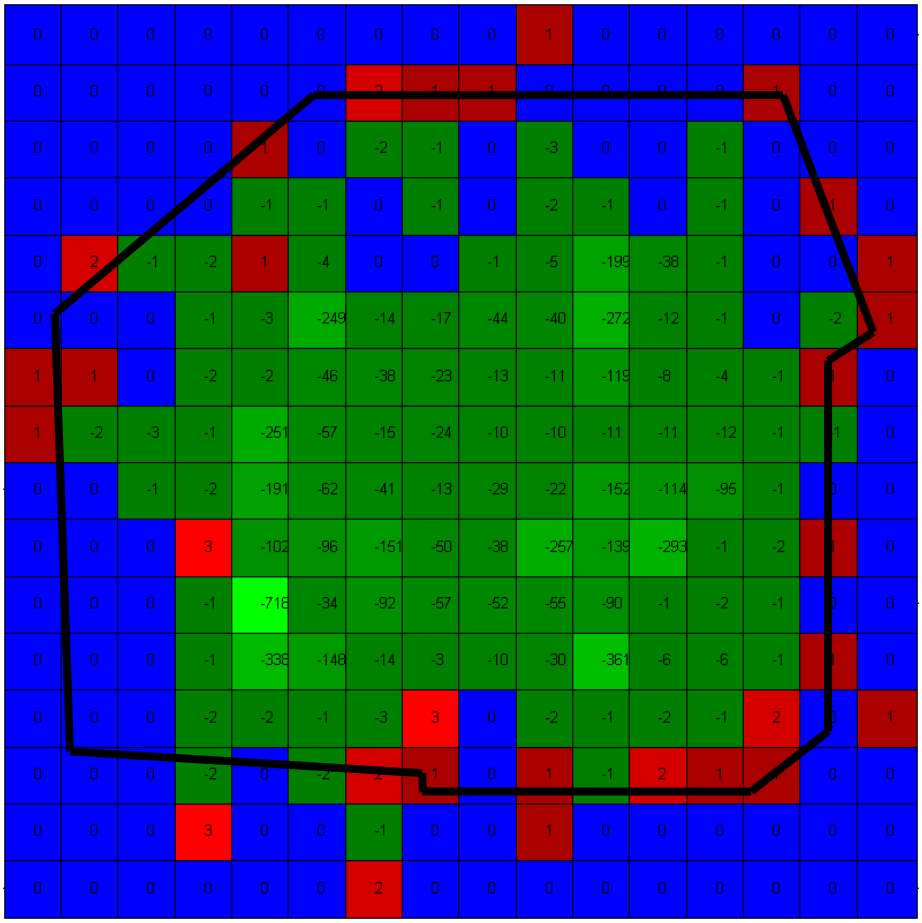

Abbildung 4.9: Karte nach 8 Minuten
Die Abbildung zeigt die aus der in Abbildung 4.8 gezeigten Fahrt entstandene Karte. Hellere Farbtöne stehen für eine höhere Wahrscheinlichkeit, die auch anhand der eingetragenen Werte pro Rasterzelle ermittelt wurde. Grün = frei, blau = unbestimmt/unsicher, rot = belegt. Die Sollgrenzen sind in schwarz dargestellt.

Abbildung 4.9 zeigt schließlich eine Karte des Testgebiets mit unterschiedlich gefärbten Kacheln, die, wie beschrieben, die Rasterzellen repräsentieren und mit dem Wahrscheinlichkeitswert ihrer Belegung beschriftet sind. Einerseits sind freie Zellen mit sehr hohen Wahrscheinlichkeiten gekennzeichnet, andererseits gibt es im Randbereich kaum statistisch sicher belegte Zellen und viele unsichere. Die zuvor beobachtete Häufung von Roboterposen in Ecken findet sich auch hier in Form besonders großer Werte wieder. Denn der Roboter addiert während seines Aufenthaltes den entsprechenden Wert. Der Wert 0 kann dabei nicht nur bedeuten, dass diese Zelle nicht erfasst wurde, sondern auch, dass konkurrierende Messungen stattgefunden haben: Beispielsweise kann von einer Roboterpose aus per Ultraschall gemessen worden sein, dass kein Hindernis in der Zelle ist, während von einer weiteren ein Kreisbogen in diese Zelle fällt. Dies erklärt auch, warum isolierte belegte Zellen vorkommen, die von scheinbar unbestimmten umgeben sind. Dies erscheint auf den ersten Blick fehlerhaft, da im Zuge der Distanzmessung alle Zellen zu einem detektierten Ziel als frei markiert werden.

Zu beobachten ist außerdem, dass der am häufigsten besuchte Bereich links unten die tatsächliche Gegebenheit, eine nahezu rechtwinklige Ecke, nicht richtig wieder gibt. Hier könnten durch die glatten Oberflächen an dieser Stelle Spiegelungen des Signals aufgetreten sein, die das Ergebnis zusätzlich verfälscht haben. Der von Pappkartons begrenzte Bereich unten rechts wird etwas besser wieder gegeben, oben rechts wurden trotz der langen Messdauer von acht Minuten zu wenige Messungen durchgeführt, was auf die zufällige Fahrstrategie zurückfällt.

Das Ungleichgewicht von hohen Werten für freie Zellen zu niedrigen für belegte, zeigt einerseits, dass die entsprechende Gewichtung, aber auch die Strategie, noch nicht ausgereift sind. Beispielsweise könnte die Markierung der Roboterpose als frei nicht zeitlich, sondern räumlich gesteuert werden, um Zellen nur einmal als frei zu markieren, die der Roboter durchfährt. Die niedrigen positiven Belegtheitszustände sind andererseits jedoch auch Ausdruck der geringen Aussagekraft des Ultraschallsensors gegenüber der sicheren Annahme, die Pose des Roboters sei frei bzw. der Zustand des Tastsensors sei sicherer als dessen Messungen. Letzterer löste bei Tests selten und vorrangig dann aus, wenn der Roboter so schräg gegen eine Wand fuhr, dass diese per Ultraschall nicht detektiert wurde. Eine solche Messung würde eine höhere Relevanz und Zuverlässigkeit bekommen, wenn zwei Tastsensoren zum Einsatz kommen, und damit bekannt ist, ob eine Zelle rechts oder links des Roboters als belegt zu markieren ist. Aber auch dann kommt der Distanzmessung die Hauptaufgabe zu. Hier hätte ein besserer Sensor den größten Effekt auf das Ergebnis.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Das in dieser Arbeit behandelte Projekt zeigt Möglichkeiten und Herausforderungen auf, die ein Zusammenspiel von Videoüberwachung und Roboterplattform mit sich bringt. Es wurde ein komplett funktionsfähiger Ablauf, an dessen Ende die Kartierung der Umwelt steht, umgesetzt. Damit stehen Daten bereit, die im Sinne der Motivation dieser Arbeit für weitere Schritte der Videoüberwachung, wie die Nutzung als Verdeckungs- und Tiefenkarte, dienen können. Insbesondere die rudimentäre Umsetzung der Kartierung mithilfe der Sensoren eines Lego-Bausatzes lässt viel Raum für Erweiterungen und Verbesserungen. Im Vordergrund steht die Umsetzung eines Testszenarios, um die verschiedensten Fragestellungen abzudecken. Gelöste Aufgaben sind die Berechnung der 3D-Roboterpose mit einer Überwachungskamera, die Kalibrierung derselben mit einem Verfahren, dessen Automatisierung mittels Roboter denkbar ist, sowie die Verknüpfung dieser Daten mit der Roboterplattform. Dies geschieht über ein Netzwerk, in dem die Frage der Zeitsynchronisation über Zeitabgleiche und die Verwendung von Zeitstempeln gelöst wurde.

Es wurde außerdem eine Roboterplattform entwickelt, die mittels einer dafür entworfenen Fahrstrategie autonom ihre Umgebung erkundet und sich per Odometrie selbst lokalisiert. Dabei wird ein Differentialantrieb verwendet, der ein einfaches Modell und gleichzeitig ein stabiles Balanceverhalten bietet und damit flexibel einsetzbar ist. Es kommen mit einem aktiven Distanzmesser und einem passiven Tastsensor die beiden Hauptklassen von exterozeptiven Sensoren zum Einsatz. Außerdem findet sich als Beispiel für einen propriozeptiven Sensor der in den Motor integrierte Drehgeber. Die Klasse der Aktuatoren wird durch die Motoren repräsentiert. Damit sind Hauptelemente der Robotik vertreten.

Der umgesetzte Ablauf deckt sämtliche Transformationen zwischen verschiedenen benötigten Koordinatensystemen ab. Insbesondere wurde ein Verfahren entwickelt und erfolgreich umgesetzt, dass automatisch das Drehzentrum des Roboters mithilfe der Beobachtungen der Überwachungskamera per Ausgleichung ermittelt.

Der gesamte Ablauf umfasst das Herstellen der äußeren Orientierung zwischen der Kamera und dem im Roboter gelagerten Objektkoordinatensystem und läuft ohne manuelles Eingreifen ab. Er liefert als Ergebnis eine Karte, die es ermöglicht, wahrscheinlichkeitsbasiert freie Gebiete von Objekten zu unterscheiden.

Die Anwendungsmöglichkeiten gehen, wie in der Einleitung der Arbeit erläutert, über den Nutzen für die Kameraüberwachung hinaus. Das Szenario ist auch für den Fall denkbar, Roboter zu unterstützen. Dies geschieht hier genau genommen schon durch die Kombination der durch die Kamera redundant zu der Odometrie gemessenen Pose des Roboters.

5.2 Ausblick

Hinsichtlich eines Ausblicks dieser Arbeit finden sich in jedem durch das Projekt tangierten Bereich Fragestellungen und Möglichkeiten zu dessen Verbesserung und Ausbau.

Wie in Kapitel 3.3 angedeutet, gibt es alternative Signalisierungsmöglichkeiten neben einem planaren Schachbrettmuster. Insbesondere um die Signalisierung über einen größeren Maßstabsbereich zu ermöglichen und die benötigte Größe, welche die des Roboters maßgeblich beeinflusst, zu verringern, sind zum Beispiel Möglichkeiten einer aktiven Signalisierung zu testen. Auch die hier eingesetzte Signalisierung ist ausbaufähig: Bei der Schachbrettdetektion berücksichtigt OpenCV die einzelnen Punkte nicht im Sinne einer bedingten Ausgleichung um die optimale Position des Schachbretts als Ganzes zu schätzen. Dies wäre durch die Geometrie des Musters möglich und es ist vor allem hinsichtlich der Definition des Schachbrettkoordinatensystems über die drei Eckpunkte von einem Mehrwert auszugehen.

Die gewählte Vereinfachung, die unterschiedlichen Schachbrettsysteme aller Epochen über eine Homographie in einem System zu vereinen, führt zwar dazu, dass im Zuge der Homographie-Schätzung in Epoche 0 ein Ausgleichungsverfahren alle Schachbrettpunkte berücksichtigt. Jedoch ist auch hier die Ausgangslage nicht optimal, um mit diesem einen Schachbrett die Bodenebene abzuschätzen. Da es vermutlich nicht genau parallel zu dieser sein wird, ergibt sich ein Fehler in den folgenden Epochen, deren Punkte nicht genau in dieser Ebene liegen. Dies ließe sich vermeiden, indem mehrere über das Testgebiet verteilte Schachbrettmessungen per Ebenenausgleichung für die Berechnung der Ebene herangezogen werden, die der Homographie zugrunde liegt. Da dies wegen des unbekannten Geländes nicht direkt möglich ist, könnte die Homographie iterativ während der Erkundung der Umgebung durch den Roboter im Sinne einer sequentiellen Ausgleichung aktualisiert werden.

Für die Bildmessung werden die Vorteile der Bildsequenz gegenüber Einzelbildern nicht ausgenutzt. Die Suche nach dem Schachbrett erfolgt in jedem Bild ohne Berücksichtigung der vorausgegangenen Position. Diese könnte jedoch als Näherungswert in die Detektion einfließen und sie durch eine Einschränkung des Suchbereichs beschleunigen. Die Schachbrettdetektion erscheint auch deshalb noch nicht ausgereizt, weil das Schachbrett sobald eine Ecke verdeckt ist, nicht mehr erkannt wird. Außerdem ist die Geschwindigkeit des OpenCV-Algorithmus für eine Videoauswertung mit wenigen Bildern pro Sekunde gering.

Bezüglich der Übertragung des Maßstabs vom Schachbrett auf die Szene, erscheint diese als nicht optimal, da die Ausmaße der Szene die des als Maßstab verwendeten Schachbretts um ein Vielfaches überschreiten. Hier ist denkbar, dass der Roboter eine möglichst lange Strecke im Bild zurück legt, diese selbst bestimmt und das detektierte Schachbrett an deren Anfang und Ende eine entsprechend lange Maßstabsstrecke liefert. Dem steht die bisher niedrige Genauigkeit der Odometrie jedoch entgegen.

Was die verwendete Kamera angeht, fällt auf, dass deren PTZ-Funktionalität nicht ausgenutzt wurde. Das bedeutet, dass die Möglichkeit, den Roboter über einen größeren Maßstabsbereich zu verfolgen in Zukunft noch genutzt werden kann. Dass dies nicht geschieht, liegt an der Vereinfachung durch eine konstante Geometrie und dem dadurch möglichen festen Kamerakoordinatensystem als Objektkoordinatensystem. Ohne diese Vereinfachung würde sich die äußere Orientierung der Kamera hinsichtlich der Bodenebene ändern. Außerdem ist die Kalibrierung für sämtliche Zoomstufen und Ausrichtungen der Kamera bisher nicht gelöst. Hinsichtlich der Kamerakalibrierung ist ein nächster denkbarer Schritt diese direkt aus dem Videostrom durchzuführen, anstatt von Hand geeignete Einzelbilder abzuspeichern. Die Bewegung der Kalibriertafel könnte ein Roboter automatisieren. Ein weiterer Schritt ist ebenfalls die Integration mehrerer Kameras, deren Sichtbereiche sich ergänzen, und die Ermittlung deren innerer sowie äußerer Orientierung mithilfe eines Roboters bzw. die Positionierung eines Roboters bei bekannten Kameraorientierungen.

Was die Roboterplattform angeht, zeichnet sich insbesondere die Frage nach dem geeigneten Antrieb durch viele Variablen und Alternativen aus, die es abzuwägen und weiter zu untersuchen gilt. Daneben zeigt sich, dass eine solide Bauweise nötig ist. Vor allem auch für die Kopplung mit der Kamera über eine Signalisierung in Form einer Signalisierungstafel. Neben der Integration weiterer und genauerer Sensoren ist gerade aus Sicht der Geodäsie hinsichtlich der statistisch robusten Auswertung der Sensordaten inklusive der Behandlung von Ausreißern viel Potenzial vorhanden. Eine umfassendere Untersuchung der Fehlerquellen jedes einzelnen Sensors und deren Modellierung, sowie in deren Zusammenspiel, ist dafür unabdingbar und konnte hier nur teilweise behandelt werden.

Liegen solche statistischen Daten vor, ergänzen diese die Messwerte für Verfahren wie die Kalman-Filterung oder *Simultaneous Localization And Mapping* (SLAM), die die Einsetzbarkeit des getesteten Szenarios deutlich erweitern würden. Denn die Kombination verschiedenster Beobachtungsquellen spielt hier eine große Rolle und die Nutzung bereits erfasster Umgebungsdaten ist für eine wesentlich effizientere Fahrstrategie unabdingbar.

Ist von genaueren Sensoren die Rede, darf abschließend nicht unerwähnt bleiben, dass außerdem ein stärkerer Rechenkern dem Roboter zu mehr Autonomie verhelfen würde: Das Ziel sollte darin liegen, in Zukunft alle Berechnungen von dem PC auf den Roboter zu verlagern und diesen direkt mit der Kamera und möglichen weiteren externen Datenquellen kommunizieren zu lassen, um ein komplett selbständiges Agieren und Reagieren auf Ereignisse in der Umwelt zu realisieren.

Literaturverzeichnis

Axis Communications. (2012). *Axis 214 PTZ Netzwerk-Kamera - Datenblatt.* Abgerufen am 19. März 2012 von http://www.axis.com/de/files/datasheet/ds_214ptz_36255_de_0907_lo.pdf

Berns, K. & Schmidt, D. (2010). *Programmierung mit LEGO MINDSTORMS NXT.* Heidelberg: Springer.

Boogarts, M., Davis, B. L., Daudelin, et al. (2007). *The Lego Mindstorms NXT Idea Book.* San Francisco: No Starch Press.

Bradski, G. & Kaehler, A. (2008). *Lerning OpenCV.* Sebastopol: O'Reilly Media.

Bredenfeld, A., Leimbach, T., Breuer, T., Trella, S. & Arnold, M. (2010). *Roberta - Programmieren mit Java: Band 3 - NXT* (2. Ausg.). Stuttgart: Fraunhofer IRB Verlag.

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Systems Journal Vol. 4, Nr. 1* , S. 25-30.

Frischknecht, C. & Other, T. (2006). *Semesterarbeit: LEGO Mindstorms NXT - Next Generation.* Zürich: ETH.

Hartley, R. & Zisserman, A. (2003). *Multiple View Geometry in Computer Vision* (2. Ausg.). Cambridge: Cambridge University Press.

Laganière, R. (2011). *OpenCV 2 Computer Vision Application Programming Cookbook.* Birmingham: Packt Publishing.

LEGO Group. (2006). *LEGO MINDSTORMS Bedienungsanleitung.*

Leimbach, T. & Trella, S. (September 2010). *LEGO Mindstorms NXT: Programmiersprachen im Überblick.* Abgerufen am 3. März 2012 von Roberta Projekthomepage: http://www.roberta-home.de/de/was-bietet-roberta/roberta-reihe/vergleich-programmiersprachen-lego-mindstorms-nxt

Luhmann, T. (2010). *Nahbereichsphotogrammetrie: Grundlagen, Methoden und Anwendungen* (3. Ausg.). Berlin: Wichmann.

Marsiske, H.-A. (2. Januar 2012). Autonome Vorhut - Neue Techniken bei Rettungsrobotern. *c't* , S. 72-75.

Marsiske, H.-A. (23. Mai 2011). Bedingt einsatzbereit - Wie Roboter im Katastrophenfall Rettungsmannschaften unterstützen. *c't* , S. 76-79.

Marsiske, H.-A. & Kruijff, G.-J. (2. Januar 2012). Volles Verständnis - Teamfähige Roboter sollen Rettungskräfte unterstützen. *c't* , S. 76-77.

Moral, J. A. (12. April 2009). *Develop leJOS programs Step by Step.* Abgerufen am 14. November 2011 von http://www.juanantonio.info/lejos-ebook/

Neuner, H. (2009). *Vorlesungsunterlagen Methodik der Ingenieurgeodäsie.* Hannover.

Niemeier, W. (2002). *Ausgleichungsrechnung.* Berlin, New York: de Gruyter.

Pas. (2012). *Google Play - IP Webcam*. Abgerufen am 19. März 2012 von https://play.google.com/store/apps/details?id=com.pas.webcam

Pitteway, M. L. (November 1967). Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter. *Computer Journal Vol. 10, Nr. 3* , S. 282-289.

Siciliano, B., & Khatib, O. (2008). *Springer Handbook of Robotics.* Berlin: Springer.

Wittke, M. (2011). *Dynamic Reconfiguration Methods for Active Camera Networks.*

Yilmaz, A., Javed, O., & Shah, M. (Dezember 2006). Object Tracking: A Survey. *ACM Computing Surveys, Vol. 38, Nr. 4, Artikel 13* .

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mir während der Bearbeitung dieser Arbeit zur Seite gestanden und zu deren Gelingen beigetragen haben. Mein besonderer Dank gilt Daniel Muhle für die Unterstützung bei der Einführung in das Thema, die Betreuung und die Erreichbarkeit bei Fragen. Martin Reich, Lukas Schack und Jana Schmidt danke ich für das engagierte Korrekturlesen und die gemeinsamen Aktivitäten abseits des Unialltags. Vielen Dank auch an meine Familie und Freunde für die Geduld, das Verständnis und die Unterstützung während der Arbeit und des gesamten Studiums. Ohne das mir entgegengebrachte Vertrauen in meine Fähigkeiten und die Unterstützung des von mir gewählten Weges wäre das Studium in dieser Form nicht möglich gewesen.

Anhang

Codeübersicht

Java:

- *nuur-common*: Definition der Kommunikationsschnittstelle
  - *nuur.common.comm*
    - *ProtocolData*
    - *ProtocolDataListener*
    - *ProtocolHandler*
    - *ProtocolFactory*
    - *MapProtocol*: Protokolldefinition
    - *OrientationProtocol*: Protokolldefinition
    - *TimeStampProtocol*: Protokolldefinition
- *PatternRobot*: Code auf dem NXT-Baustein
  - *patternrobot.nxt.comm*: Prozess für die Kommunikation
    - *BTCommThread*
    - *NXTConnectionHandler*
  - *patternrobot.nxt.main*
    - *OdometryThread*: Eigenständiger Prozess für die Odometrieberechnungen
    - *PatternRobot*: Hauptprogramm des Roboters mit Fahrstrategie, Hinderniserkennung und Kartierung
- *PatternRobotPC*: Code auf dem PC
  - *patternrobot.pc.comm*
    - *BTComm*
    - *BTCommThread*: Kommunikationsprozess
    - *OriReader*: Verzeichnisüberwachung des C++-Codes
    - *PCConnectionHandler*
    - *ProtocolOperator*
    - *TimeAndFilewriterThread*: Zeitabgleich zwischen PC und NXT sowie Schreiben von Log-Dateien
  - *Patternrobot.pc.map*: Implementierung der Karte
    - *GridMap*: Hauptklasse inklusive Odometrie-Kamera-Kombination und Kartenaktualisierung mithilfe der verschiedenen Beobachtungstypen
    - *GridPoint*: Repräsentation einer Rasterkoordinate
    - *GridSquare*: Inhalt einer Rasterkoordinate (Belegtheitszustand)

C++:

- *robottracking_cam.cpp*
  - *runCalibration*: Umsetzung der Kamerakalibrierung
  - *runTracking*: Umgang mit Videostrom, gesamtes Tracking des Roboters inklusive Berechnung der Homographie, der Transformationen und kontinuierlichem Schreiben der Positionsdateien für den Java-Code

- *circlefit*: Ausgleichung für die Kalibrierung des Drehzentrums
- *chessDist*: Distanzmessung mithilfe eines Schachbretts

Matlab:

- *analysis.m*: Ebenenschätzung in 3D-Schachbrett-Punktwolke
- *bildkoordGenauigkeit.m*: Berechnung statistischer Daten der wiederholten Bildmessungen
- *camOdoDiffs.m*: Plotten von Histogrammen der Differenzen der einzelnen Kartenaktualisierungen aus Odometrie und Kamera
- *camOdoVisualizer.m*: Grafische Ausgabe der Tracks von Kamera und Roboter (Odometrie)
- *robot_out.m*: Plotten der Rasterkarte aus HashMap

Daten-CD

Der Arbeit ist eine Daten-CD beigelegt, auf der sich, neben dieser Ausarbeitung, sämtliche Quellcodes, Analysetabellen, Abbildungen, Fotos, Screenshots und Ausgabedaten finden. Die Quellcodes sind kommentiert, das Java-Doc des in Java programmierten Teils, liegt ebenfalls bei.