Institut für Mensch-Maschine-Interaktion

RWTH AACHEN UNIVERSITY

SLAM-basierte Autonome Indoor-Navigation für mobile Roboter

Masterarbeit

von
B.Sc. Johnson Luo Loh

Diese Arbeit wurde vorgelegt am Institut für Mensch-Maschine-Interaktion.

Prüfer: Univ.-Prof. Dr.-Ing. Jürgen Roßmann

Aachen, 07. Mai. 2018

SLAM-basierte Autonome Indoor-Navigation für mobile Roboter

SLAM-based Autonomous Indoor-Navigation for mobile Robots

von
B.Sc. Johnson Luo Loh
318755

Betreuer:
Dipl.-Inform. Björn Sondermann

Kurzfassung

Die sichere Navigation zu einer Position im Raum ist eine Hauptaufgabe im Bereich der mobilen Robotik. Dazu ist es notwendig die Lage des Roboters zu bestimmen und sich kollisionsfrei zum Ziel zu bewegen. Besonders in Umgebungen mit engen Passagen und Büroumgebungen muss die sichere Navigation des Roboters für sich und seine Umgebung gewährleistet werden.

In dieser Arbeit werden Lokalisierungs- und Navigationsverfahren für die Roboterplattform *SeekurJr* vorgestellt. Die Lokalisierung wird durch marker-basierte SLAM-Algorithmen realisiert, um die Pose (Position und Orientierung) des Roboters zu tracken. Für die Navigation wird der *Dynamic Window Approach* zur Kollisionsvermeidung verwendet. Die Verfahren sollen für Systeme mit Differentialantrieb anwendbar sein. Die Kollisionsvermeidung soll für Roboter mit einem rechteckigen Umriss konzipiert werden.

Das Simulations- und Visualisierungssystem VEROSIM des Institut für Mensch-Maschine-Interaktion der Rheinisch–Westfälische Technische Hochschule wird als Interface für den Roboter verwendet sowohl in der Simulation als auch am realen Prototypen. Ziel der Arbeit ist es die Funktionalitäten in einer möglichst modularen Struktur zu implementieren, sodass diese für eine Vielzahl von Situationen anwendbar und für alternative Verfahren erweiterbar sind.

Inhaltsverzeichnis

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt Univ.-Prof. Dr. Ing. Jürgen Rossmann mein Dank für die Möglichkeit meine Masterarbeit am Institut für Mensch und Maschine Interaktion durchzuführen. Ebenfalls möchte ich mich bei meinem Betreuer Dipl.-Ing. Björn Sondermann für die hilfreichen Anregungen und die konstruktive Kritik bedanken.

Ein besonderer Dank gilt meiner Mutter Vi Phan Loh und meiner Schwester Karin Loh, die mich bei meinem Studium motiviert, unterstützt und einen Leitfaden für diese Zeit gegeben haben.

Abschließend möchte ich mich bei meinen Freunden bedanken, die mit ihrer Einsatzbereitschaft und interessanten Beiträgen dazu beigetragen haben, dass diese Masterarbeit in dieser Form vorliegt.

Verzeichnisse

Abbildungen

Tabellen

Dateiauszüge

Akronyme

AR	Augmented Reality
MMI	Institut für Mensch-Maschine-Interaktion
QR	Quick Response
RWTH	Rheinisch–Westfälische Technische Hochschule
FTS	Fahrerlose Transportsysteme 1
BIM	Building Information Model 2
DWA	Dynamic Window Approach 3
EKF	Extended Kalman Filter 10
UKF	Unscented Kalman Filter 10
SLAM	Simultaneous Localization and Mapping 12
LOS	line of sight 39
SDK	Software Development Toolkit 50

Nomenklatur

Die in dieser Arbeit verwendete Notation mathematischer Symbole und die verwendeten Formelzeichen sind im Folgenden zusammengefasst.

$s \in \mathbb{R}$	Allgemeiner Skalar
$\boldsymbol{b} \in \mathbb{R}^n$	Allgemeiner Vektor der Dimension $n \in \mathbb{N}$
$b_i \in \mathbb{R}$	i-te Komponente des Vektors $\boldsymbol{b} \in \mathbb{R}^n$ mit $0 \leq i < n \in \mathbb{N}$
$\boldsymbol{b}(t) \in \mathbb{R}^n$	Allgemeiner Vektor als Funktion der Zeit $t \in \mathbb{R}$
$\dot{\boldsymbol{b}}(t) \in \mathbb{R}^n$	Erste Ableitung eines allgemeinen Vektors nach der Zeit
$\mathbf{A} \in \mathbb{R}^{m \times n}$	Allgemeine Matrix mit $n \in \mathbb{N}$ Zeilen und $m \in \mathbb{N}$ Spalten
$\mathbf{A}^T \in \mathbb{R}^{m \times n}$	Transponierte einer Matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$
$\mathbf{A}^{-1} \in \mathbb{R}^{m \times m}$	Inverse einer quadratischen Matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$
$\mathbf{x}_r \in \mathbb{R}^3$	Konfiguration des Roboters in 2D
$\mathbf{u}_{speed} \in \mathbb{R}^2$	Odometrie des Roboters als Geschwindigkeitstupel
$\mathbf{u} \in \mathbb{R}^2$	Odometrie des Roboters als Streckentupel
$\mathbf{x}_r \in \mathbb{R}^3$	Konfiguration des Roboters in 2D
$\mathbf{x}_m \in \mathbb{R}^3$	Konfiguration eines Markers in 2D
$\mathbf{I} \in \mathbb{R}^{N \times N}$	Einheitsmatrix
$\mathbf{x} \in \mathbb{R}^N$	Zustand des Kalman Filters
$\mathbf{P} \in \mathbb{R}^{N \times N}$	Kovarianzmatrix des Kalman Filters
$\mathbf{z} \in \mathbb{R}^M$	Messvektor
$\mathbf{Q} \in \mathbb{R}^{3 \times 3}$	Systemrauschen
$\mathbf{R} \in \mathbb{R}^{3 \times 3}$	Messrauschen
$\mathbf{J}, \mathbf{H}, \mathbf{L} \in \mathbb{R}^{m \times n}$	Jakobimatrizen
$\mathbf{K} \in \mathbb{R}^{N \times M}$	Kalmangain
$\mathbf{P}_{yy}, \mathbf{P}_{xy}$	Auto- und Kreuzkorrelationsmatrizen

$v_r, v_l \in \mathbb{R}$	Tangentialgeschwindigkeit des rechten und linken Rads auf ihrem Radumfang
$d_{rad} \in \mathbb{R}$	Raddurchmesser
$L \in \mathbb{R}$	Radabstand
$l_{robot} \in \mathbb{R}$	Roboterlänge
$w_{robot} \in \mathbb{R}$	Roboterbreite
$R \in \mathbb{R}$	x-Komponente des Rotationszentrums einer radialen Bewegung im Roboterkoordinatensystem
$T \in \mathbb{R}^{3\times3}$	Transformationsmatrix
$d_{kante} \in \mathbb{R}$	Kantenlänge eines Markers
$\epsilon_{kante}, \epsilon_{verzerrung}$	Posenfilterparameter
$\epsilon, \epsilon_{front}$	Parameter für Sicherheitsabstände
p	Allgemeines Symbol für Wahrscheinlichkeiten
$\mathbf{p}_{i,coll} \in \mathbb{R}^2$	i-ter Punkt der Kollisionspunktwolke
$\|\|.\|\|$	Euklidische Norm
$arc(R, \mathbf{p}_{i,coll})$	Maximale Pfadlänge des Roboters auf dem radialen Pfad mit Radius R zum Punkt $\mathbf{p}_{i,coll}$
$\overset{\frown}{AB}$	Kreisbogenlänge von A nach B
$G, head, clear, vel$	Gewichtsfunktionen des Dynamic Window Approachs

KAPITEL 1

Einführung

Dieses Kapitel dient zur Einführung der in dieser Arbeit adressierten Problematik. Dazu werden im ersten Teil Anwendungsmöglichkeiten sowie typische Rahmenbedingungen skizziert. Im zweiten Teil wird eine Lösungsmöglichkeit für die Problematik erläutert, die in dieser Arbeit verfolgt wird.

1.1 Motivation

Im Zeitalter der Digitalisierung werden mobile autonome Systeme zunehmend in die Gesellschaft integriert. Diese sollen vor allem wiederkehrende Prozesse übernehmen, um die Produktivität zu steigern. Dazu ist es nötig die Position dieser Systeme im Raum zu bestimmen und kollisionsfrei zu navigieren. Da es dabei unweigerlich zum Kontakt zwischen Mensch und Maschine kommen kann, müssen die Systeme so ausgelegt sein, dass sie weder die Umgebung, noch sich selbst, gefährden.

Anwendungen für mobile Roboter finden sich vor allem in der Industrie, welche sich mit der Idee der „Industrie 4.0“ verbinden lassen. In intelligenten und digital vernetzten Systemen stellen diese mobilen Platformen die Aktorik und die Schnittstelle zur realen Welt dar. Zum Beispiel werden Fahrerlose Transportsysteme (FTS) in kontrollierten Umgebungen eingesetzt, um die Herstellung von Fahrzeugen zu automatisieren (Abb. 1.1).

Für Verbraucher werden vollautomatisch navigierende Systeme immer attraktiver. Unter anderem können sie im Haushalt alltägliche Aufgaben ausführen. Eine gezielte kollisionsfreie Bewegung ist für komplexe Systeme wie Serviceroboter essentiell. Wenn dies gewährleistet ist, können in Zukunft Roboterassistenten im menschlichen Umfeld eingesetzt werden (Abb. 1.2).

Die Grundstruktur der meisten Einsatzgebiete innerhalb von Gebäuden sind vorher bekannt und können entsprechend den Anforderungen umgestaltet werden. Über Gebäu-

Abbildung 1.1: *Fahrerlose Transportsysteme der Audi R8 Produktionsanlage [Med18].*

Abbildung 1.2: *Serviceroboter des Fraunhofer Instituts für Produktionstechnik und Automatisierung unter dem Produktnamen „Care-O-bot“ in Interaktion für den Haushalt [IPA18].*

dedaten wie das Building Information Model (BIM) sind die Grundrisse von Räumen klar definiert und stellen Informationen über die Umgebung bereit, die für die Lokalisierung und Navigation von autonomen Plattformen verwendet werden können. Insbesondere sind im vornherein angebrachte Landmarken für die Lokalisierung interessant, da diese in ihrer Position und Ausrichtung statisch sind. Daher wird in dieser Arbeit angenommen, dass sich solche Informationen[1] in einer geeigneten Form, wie zum Beispiel über QR-Codes am Eingang eines Gebäudes, bereitgestellt werden.

Auch in konstruktionstechnischer Hinsicht können mobile Roboter unterschiedliche For-

[1] In dieser Arbeit wurden AR-Marker verwendet (siehe Kap. 3.1)

men annehmen, von runden Staubsaugerrobotern bis hin zu autonomen Radladern. Nicht nur die Form, sondern auch die Art und Weise wie die Antriebe angebracht sind, können den Bewegungsfreiraum einschränken. Diese sogenannten *non-holonomic constraints* sind bei den meisten Bauweisen üblich und können in der Modellierung berücksichtigt werden, um das maximale Potential des Systems auszunutzen. Ein Beispiel hierfür ist ein autonom parkendes Auto: Wäre die Bewegung des Autos in allen Freiheitsgraden möglich, dann kann sich das Auto senkrecht zu der gewohnten Fahrtrichtung bewegen und einparken. Stattdessen muss ein Fahrmanöver durchgeführt werden.

1.2 Ziel der Arbeit

Angesicht des Aufgabenfelds müssen zwei Hauptaufgaben bewältigt werden: Die Lokalisierung und die Navigation.

Aufgrund des Aufbaus vom Seekur Jr soll bei der Lokalisierung ein markerbasierter SLAM-Algorithmus verwendet werden, der auf Basis eines Bewegungsmodells und Odometriedaten iterativ eine Pose (Position und Orientierung) prädiziert und anhand von Messdaten der Marker diese Pose korrigiert. Die in dieser Arbeit verwendeten visuellen Marker werden über ein Kamerasystem erfasst, das an dem Roboter angebracht ist. Dabei sollen sowohl Landmarken berücksichtigt werden, die vorher bekannt sind, als auch neu angebrachte Marker kartiert und für die Lokalisierung mitverwendet werden.

Die sichere Navigation soll mittels dem Verfahren *Dynamic Window Approach (DWA)* zur Kollisionsvermeidung abgesichert werden. Dabei soll der ursprüngliche Algorithmus[2] an recheckige Robotergrundrisse angepasst werden. Weiterhin soll die Kollisionsvermeidung Daten von einem Laserscanner verwenden, der am Roboter angebracht ist. Da ein Sensor für eine komplette Abdeckung des Roboters nicht ausreicht, sollen die Sensordaten in einer Rasterkarte kartiert werden, um Informationen im toten Winkel des Lasersensors zu ersetzen.

Die Umgebungen, in der die Plattform navigieren soll, sind in dieser Arbeit für Gebiete im Inneren des Gebäudes konzipiert. Dabei wird die Bewegung von einem Stockwerk zum anderen vernachlässigt und es wird sich dabei auf eine ebene Bewegung des Roboters beschränkt. Diese Arbeit konzentriert sich auf die mobile Roboterplatform *Seekur Jr* und implementiert zunächst nur das Bewegungsmodell und die Form dieser Roboterklasse. Die dabei verfügbaren Sensoren sind ein Lidarsensor und mehrere Webcams.

Diese Methoden werden in der Softwareumgebung *VEROSIM* modular implementiert, sodass die zunächst auf den Seekur Jr zugeschnittenen Algorithmen auf eine Vielzahl an Robotern und Szenarien anwendbar sind. Desweiteren sollen auch alternative Methoden

[2] aus [Die97]

der Lokalisierung und Navigation in Kombination miteinander verwendet werden können.

KAPITEL 2

Theoretische Grundlagen

Dieses Kapitel befasst sich mit den Grundlagen, die für die vorliegende Problematik benötigt werden: die Modellierung, die Lokalisierung und die Navigation eines mobilen Roboters. In Fragestellungen ausgedrückt:

- Wie verhält sich das System? Wie kann das System beeinflusst werden?
- Wo befindet sich das System in der gegebenen Umgebung? Wie schaut die gegebene Umgebung aus?
- Wie wird das System von einem Punkt zum Anderen bewegt? Wie wird eine Kollision vermieden?

2.1 Modellbildung

Die Algorithmen aus Kapitel 2.2 und 2.3 setzen voraus, dass das Systemverhalten des Roboters modelliert werden kann und somit eine Prädiktion des Verhaltens auf bestimmte Eingänge in das System möglich ist. Daher ist eine hinreichend genaue Approximation des Systems durch ein mathematisches Modell notwendig.

Die Bewegungsplattform des *Seekur Jrs* ist vergleichbar mit dem eines Panzer, welches jeweils zwei Räder auf einer Seite besitzt, die unabhängig voneinander rotieren können. Daher sind Rotationen auf der Stelle sowie Vor- und Rückwärtsbewegungen möglich, sofern an den Motoren eine konstante Eingangsspannung anliegt. Dabei wird davon ausgegangen, dass die zwei Räder auf einer Seite des Roboters synchronisiert sind und somit insgesamt eine Rotationsgeschwindigkeit pro Seite beeinflussbar ist. Die resultierende Bewegung ist entweder geradlinig oder kreisbogenförmig (siehe Abb. 2.1). Eine Bewegung seitlich zur Orientierungsrichtung erfordert ein Manöver[1].

[1] Eine Serie aus konstanten Eingangssignalen

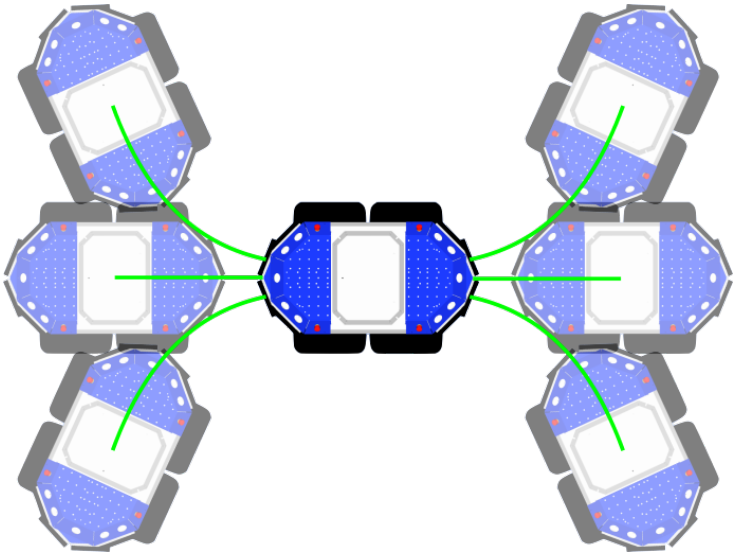


Abbildung 2.1: *Visualisierung der Bewegungsfreiheitsgrade des Seekur Jrs für konstante Eingangsspannungen der Antriebsmotoren.*

Diese Art der Fortbewegung wird im Englischen als *skid steering* bezeichnet und ist ein Variante des Differentialantriebs (*differential drive*) mit mehreren Rädern. Durch die Betrachtung des Roboters als Pose in seinem Rotationsschwerpunkt, ist seine Konfiguration im zweidimensionalen kartesischen Koordinatensystem eindeutig als Tupel $\mathbf{x}_r = (x_r, y_r, \theta_r)^T$ definiert, wobei x_r und y_r die kartesischen Koordinaten und θ_r die Orientierung sind. Zur Modellierung des Systems wird das **Unicycle Model** verwendet, der die Bewegung vereinfacht als ein Einrad beschreibt. Infolgedessen sind die Konfigurationsübergangsgleichungen wie folgt definiert:

$$\begin{aligned} \dot{x_r} &= u_v \cdot \cos(\theta_r) \\ \dot{y_r} &= u_v \cdot \sin(\theta_r) \\ \dot{\theta} &= u_\omega \end{aligned} \tag{2.1}$$

Dabei notieren u_v und u_ω die Kontrollvariablen für Translation- und Rotationsgeschwindigkeit. Im Vektor ergeben sie $\mathbf{u}_{speed} = (u_v, u_\omega)^T$. Um dieses Modell auf den Differentialantrieb zu übertragen, müssen der Raddurchmesser d_{rad} und der Radabstand L bekannt sein. Dann lassen sich die Kontrollvariablen auch ausdrücken als:

$$\begin{aligned} u_v &= \frac{v_r + v_l}{2} \cdot \frac{d_{rad}}{2} \\ u_\omega &= \frac{v_r - v_l}{2} \cdot \frac{d_{rad}}{L} \end{aligned} \tag{2.2}$$

Es ist zu beachten, dass sich die Parameter auf den Differentialantrieb beziehen, der jeweils ein Rad auf einer Seite besitzt. Die Variablen v_r und v_l beschreiben in diesem Zusammenhang die Tangentialgeschwindigkeiten des rechten und linken Rades auf ihrem Radumfang[2]. Während sich die Übertragung auf mehrrädrige Systeme in der Translationskomponente übertragen lässt, ist die Reibung sowie die Abstände der Räder auf einer Seite untereinander bei einer Rotationsbewegung nicht zu vernachlässigen. Da sich die Reibungskraft proportional zur Geschwindigkeit verhält sowie die Radabstände symmetrisch verteilt und konstant sind, kann ihr Einfluss zusammen als Konstante $\xi \in (0,1)$ vereinfacht modelliert werden. Diese wird zu der Rotationsgeschwindigkeit in Glg. 2.2 multipliziert. Für eine detailliertere Analyse des *skid steering* wird auf [Sha99] verwiesen.

Aus geometrischer Sicht ist die Trajektorie der Roboterpose bei konstantem Kontrolleingang $\mathbf{u}_{speed}$ über einem festen Zeitraum Δt als Kreis- und Liniensegmente darstellbar. Im Koordinatensystem des Roboters, welches seine Position im Ursprung besitzt und seine Orientierungsrichtung parallel zur y-Achse ausgerichtet ist, bedeutet dies, dass folgende Beziehung gilt:

$$
{}^{R}\Gamma = \begin{cases} \{(x,y)^T : x = 0, y \in [0, u_v \cdot \Delta t]\} & , u_v \neq 0 \wedge u_\omega = 0 \\ \{(x,y)^T : \begin{pmatrix} -R\cos(t) + R \\ R\sin(t) \end{pmatrix}, t \in [0, \frac{u_v \cdot \Delta t}{R}], R = \frac{u_v}{-u_\omega}\} & , u_v \neq 0 \wedge u_\omega \neq 0 \\ \{(x,y)^T : x = 0, y = 0\} & , \text{sonst} \end{cases} \tag{2.3}
$$

Bei genauerer Betrachtung von Glg. 2.3 ist zu erkennen, dass im Fall $u_v \neq 0 \wedge u_\omega \neq 0$ der Radius R der Trajektorie sowohl positiv als auch negativ sein kann. Unter der Voraussetzung, dass $u_v > 0$ ist[3], ist die Rotationsgeschwindigkeit u_ω für die Endorientierung verantwortlich. In Abb. 2.2 ist exemplarisch eine Bewegung im Roboterkoordinatensystem dargestellt. Die Orientierungsrichtung der Rotationsgeschwindigkeit ist dabei in die mathematisch positive Richtung definiert.

Für die Repräsentation in Weltkoordinaten wird die entsprechende Kurve ${}^{R}\Gamma$ über die Roboterpose ins Weltkoordinatensystem transformiert. Die homogene Transformation im 2-D Fall ergibt sich aus der Translation $(x_r, y_r)^T$ und Rotation θ_r zu

$$
{}^{W}T_R = \begin{pmatrix} \cos(\theta_r) & -\sin(\theta_r) & x_r \\ \sin(\theta_r) & \cos(\theta_r) & y_r \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.4}
$$

Eine wichtige Anmerkung ist, dass im kontinuierlichen Fall von Glg. 2.1 die System-

[2] Alternativ können die Winkelgeschwindigkeiten verwendet werden
[3] Diese Designentscheidung wird in Kapitel 4 näher erläutert

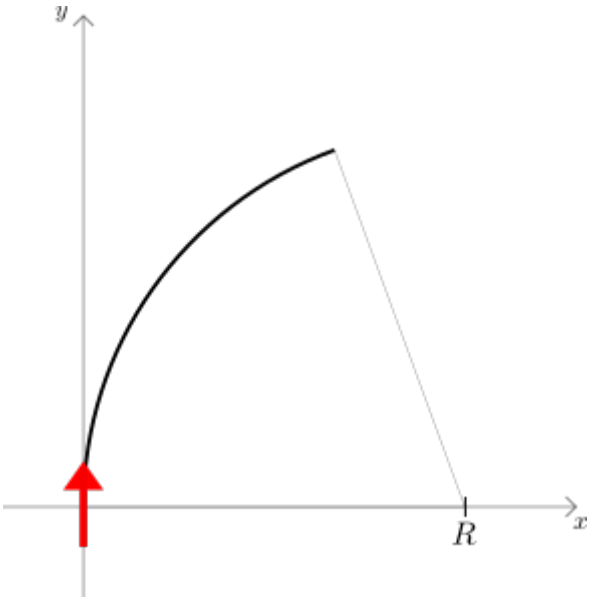


Abbildung 2.2: *Veranschaulichung einer Trajektorie für $u_v > 0$ und $u_\omega < 0$ unter Berücksichtigung der non-holonomic constraints des unicycle models.*

dynamik vernachlässigt wurde. Somit werden Einflussfaktoren wie Trägheit, Rollreibung oder begrenzte Beschleunigung im Modell ignoriert. Für diese Anwendung werden die dadurch entstehenden Modellungenauigkeiten bei der Lokalisierung als Systemrauschen berücksichtigt.

2.2 Lokalisierungsmethoden

In der Lokalisierung wird in der Literatur zwischen zwei Fällen unterschieden ([S T05, Kap. 7.2]):

- Lokale Lokalisierung:
  In diesem Fall ist die Pose zum Anfangszeitpunkt der Lokalisierung bekannt oder wird initial auf eine beliebige Pose gesetzt. Dadurch kann inkrementell die Pose geschätzt bzw. korrigiert werden. Der Fehler der Schätzung bleibt relativ klein und beschränkt sich auf die Modell- und Messungenauigkeiten.

- Globale Lokalisierung:
  Hier ist die Pose des Roboters unbekannt und muss aus den zur Verfügung stehenden Daten bestimmt werden. D.h. zum Anfangszeitpunkt kann der Schätzungsfehler sehr groß sein. Das Verfahren hierbei schränkt mit Hilfe von Umgebungsinformationen die Anzahl der Möglichkeiten für den ganzen Konfigurationsraum ein, sodass

sie schließlich gegen die tatsächliche Pose konvergiert.

In einem System bei der nur die relative Lage der Objekte zueinander entscheidend ist, liegt die Hauptproblematik in der lokalen Lokalisierung, auf welche sich diese Arbeit konzentriert.

Basierend auf dem modellierten System kann eine Posenschätzung durchgeführt werden. Dabei ist die einfachste Möglichkeit das sogenannte *dead reckoning*, in dem iterativ mit den Kontrolleingängen der nächste Zustand aus dem momentanen Zustand geschätzt wird. Mathematisch bedeutet dies, dass aus der Konfiguration zu einem Zeitpunkt $\mathbf{x}_r(t)$ mit Glg. 2.1 die Konfiguration zum nächsten Zeitpunkt $\mathbf{x}_r(t+dt)$ errechnet wird. Üblicherweise sind dabei Odometriedaten gegeben, die in eine Schätzung $\tilde{\mathbf{u}}$ vom Kontrolleingang $\mathbf{u}$ überführt werden können. Der Hauptnachteil dieser Methode ist der kumulative Fehler. D.h. jede Abweichung in $\tilde{\mathbf{u}}$ oder in der Modellierung des Systems wird in jedem Schritt aufaddiert und führt zu großen Ungenauigkeiten.

Aus diesem Grund werden weitere Sensordaten verwendet. Die Wahl der Sensoren erstreckt sich von diversen Lidar-/Radarsensoren bis hin zu optischen Sensoren, solange diese Informationen aus der Umgebung erfassen, die für die Lokalisierung verarbeitet werden können. Typischerweise sind dies Abstände zu bestimmten Landmarken oder Hindernissen. Diese Informationen werden zusammen mit dem Systemmodell in Filteralgorithmen angewendet, um die Lokalisierungsgenauigkeit zu verbessern[4]. Diese Algorithmen zielen vor allem darauf ab die Ungenauigkeiten im Systemmodell und in der Messung zu modellieren. Das Grundkonzept ist hierbei die probabilistische Darstellung des kompletten Systems von der Roboterpose bis hin zum Sensormodell.

2.2.1 Bayes Filter

Im Detail wird die (*a-posteriori*) Wahrscheinlichkeit, in der ein Zustand $\mathbf{x}_t$ auftritt, als

$$bel(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{z}_t \dots \mathbf{z}_0, \mathbf{u}_t \dots \mathbf{u}_0) \tag{2.5}$$

definiert. Dabei steht $\mathbf{z}_t$ für den Vektor aller Messungen zum Zeitpunkt t. Die Tiefstellung der Zeitvariablen t wird in diesem Abschnitt verwendet, um die Zugehörigkeit der Parameter zu bestimmten Punkten auf der Zeitachse zu beschreiben. Man erkennt, dass die bedingte Wahrscheinlichkeit abhängig von allen Kontrolleingängen und Messungen ist. Indirekt wurde hier angenommen, dass zum Zeitpunkt t die Messung $\mathbf{z}_t$ und der Eingang $\mathbf{u}_t$ beide verarbeitet wurden. Die Definition der Wahrscheinlichkeit vor Verarbeitung der

[4] siehe [S T05] für eine Übersicht

Messung (*a-priori*) ist dann

$$\overline{bel}(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{z}_{t-1} \dots \mathbf{z}_0, \mathbf{u}_t \dots \mathbf{u}_0). \tag{2.6}$$

Die Übergänge von einer Zustandswahrscheinlichkeit in die Nächste werden in den Filteralgorithmen kalkuliert. Dabei wird der Schritt von $bel(\mathbf{x}_t)$ zu $\overline{bel}(\mathbf{x}_{t+1})$ als *Prädiktionsschritt* bezeichnet. In diesem Schritt wird ausgehend vom Eingang[5] die nächste Zustandsverteilung über das Systemmodell berechnet. In den meisten Methoden wird ein additives unkorreliertes Systemrauschen modelliert, d.h. jeder Prädiktionsschritt erhöht die Unsicherheit und somit die Varianz der Wahrscheinlichkeitsverteilung. Der umgekehrte Schritt von $\overline{bel}(\mathbf{x}_{t+1})$ zu $bel(\mathbf{x}_t)$ wird hingegen als *Korrekturschritt* bezeichnet. Hierbei werden die Sensormessungen für eine Zustandskorrektur ausgewertet, wodurch die Unsicherheit des Zustands sinkt. Da die Messungen ebenfalls in diesem Filtermodell einem Rauschen unterliegen, konvergiert das System mit abwechselnder Prädiktion und Korrektur zu einer festen Varianz.

Der im oberen Abschnitt kurz skizzierte Filteralgorithmus ist der **Bayes Filter** und bildet die Basis für die probabilistischen Lokalisierungsalgorithmen [S T05]. Die populärsten Verfahren sind dabei:

- die Gaußschen Filter (z.B. Kalman-Filter)
- die nicht-parametrische Filter (z.B. Partikelfilter)

Obwohl der Partikelfilter den geringsten Schätzfehler aufweist (siehe [Sta16]), muss bei der Wahl des Algorithmus neben der Lokalisierungsgenauigkeit auch die Rechen- und Speichereffizient betrachtet werden. Auch die zur Verfügung stehenden Sensorinformationen sind dafür entscheidend. In dieser Arbeit wurden die Gaußschen Filter, speziell der Extended Kalman Filter (EKF) und der Unscented Kalman Filter (UKF), verwendet, welche in Kapitel 3.2 genauer erläutert werden. Umsetzungen von nicht-parametrischer Filter in der mobilen Robotik existieren und wurden erfolgreiche in zahlreichen Szenarien angewendet (siehe [MSW02] und [G G05]).

2.3 Pfadplanung und Kollisionsvermeidung

Die Problematik der Pfadplanung besteht darin bei bekannter Start- und Zielkonfiguration unter Berücksichtigung diverser Nebenbedingungen (u.a. Hidernissen, Roboterausdehnung, Bewegungsmodell) das Ziel möglichst optimal zu erreichen. Inwiefern der Roboter

[5] genauer die Schätzung des Eingangs auf Basis von Odometriedaten

sich optimal verhält, lässt sich an vielen Faktoren messen, wie z.B. Pfadlänge, Abfahrgeschwindigkeit des Pfades, Sicherheit, Echtzeitfähigkeit usw. . Für die Planung des Pfades gibt es auch analog zu den Lokalisierungsmethodiken (siehe Kap. 2.2) zwei Ansätze:

- Lokale Pfadplanung:
  Das System, das mit der lokalen Pfadplanung navigiert, wird auch reaktives System bezeichnet, da es nur entsprechend des momentanen Zustandes agiert. Daher nutzt dieses System auch Umgebungsinformationen aus vorherigen Zeitschritten nicht für die Planung. Offensichtlich ist der Nachteil dieser Methode, dass die gefundene Lösung nicht optimal oder eine zielführende Lösung nicht gefunden werden kann. Dennoch ist sie aufgrund ihrer geringen Rechenkomplexität besonders für Kollisionsvermeidung in hochdynamischen Umgebungen interessant. Beispiele für anfängliche Algorithmen sind zum Beispiel die Potentialfeldmethoden [Kha85].

- Globale Pfadplanung:
  Die Algorithmen der globalen Pfadplanung setzen meistens eine Form der Umgebungsrepräsentation voraus, sodass analytisch oder geometrisch eine optimale Lösung bzgl. oben genannter Metriken errechnet werden kann. Sensorinformationen müssen daher in die Umgebungsrepräsentation integriert werden, damit diese in der Pfadplanung berücksichtigt werden. Bei vielen und schnellen Änderungen in der Umgebung ist diese Methode jedoch problematisch. Eine Übersicht über aktuelle globale Planer ist in [D G16]

Ausgehend von der in Kap. 1.2 beschriebenen Anwendungsumgebung, ist es sinnvoll sich zunächst mit den **reaktiven Methoden** auseinander zu setzen. Dennoch sollten die Schwächen der reaktiven Methoden beachtet werden, da eine Vollautomation des Systems u.U. dadurch nicht möglich ist. Dabei ist die primäre Schwierigkeit das Problem der lokalen Konvergenz und nicht die der nicht-optimalen Lösung. Im Fall von Trichterformationen der Kollisionsobjekte sowie einer auf die nähere Umgebung beschränkte „Sicht“ der Sensoren kann der Roboter sich festfahren und somit sein Ziel nie erreichen.

KAPITEL 3

Marker-basierte Lokalisierung

In diesem Kapitel wird der implementierte Lokalisierungsalgorithmus näher erläutert. Dabei sollen visuelle Marker verwendet werden. In diesem Zusammenhang wird die Problematik des **Simultaneous Localization and Mapping (SLAM)** betrachtet, da die Lokalisierung nicht nur auf bekannten Markern stattfindet, sondern auch neue Marker kartieren und mit denen lokalisiert werden soll.

3.1 AR-Marker Detektion

Generell können in visuellen Markern Informationen kodiert werden, sodass diese komprimiert vorliegen und bei Bedarf wieder ausgelesen und dekodiert werden können. Im Fall, dass diese Informationen für die Lokalisierung verwendet werden, können die Informationen absolute Positionsinformationen oder Identitätsdaten enthalten. Auch ihre Form kann Aufschluss über ihre relative Lage geben.

Es werden in dieser Arbeit die ArUco *Augmented Reality Marker* ([GMM+14]) als Landmarken verwendet. Diese sind in ihrer quadratischen Form und in ihrer Kantenlänge bekannt. Innerhalb ihrer Form ist allein die ID des Markers binärkodiert. Ein Beispiel eines Markers ist in Abb. 3.1 dargestellt.

Abbildung 3.1: *ArUco AR Marker mit der ID 33.*

Um die Marker zu erfassen, ist das verwendete System mit Kameras ausgestattet, um aus den Bildern die Distanz zum Objekt, sowie ihre Ausrichtung zu errechnen. Mit dem Modell der Lochkamera ist es möglich über den Dreisatz die Distanz und die Lage der

Kanten zu bestimmen. Dazu muss die Kantenlänge, die Auflösung der Bilder sowie andere Kameraparameter bekannt sein.

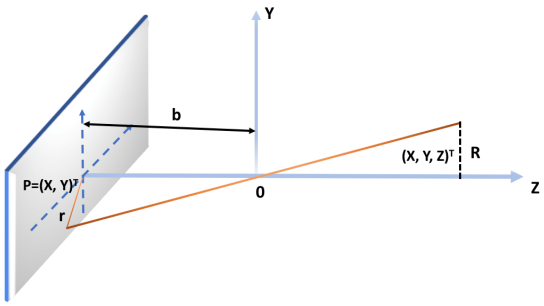


Abbildung 3.2: *Prinzip einer Lochkamera.*

Dieses theoretische Prinzip hat in der Praxis einige Probleme, die addressiert werden müssen:

- Beschränkte Auflösung:
  Die Bestimmung der Kantendistanz ist mit einem Quantisierungsfehler behaftet, da die Bildauflösung einer Kamera begrenzt ist. Der Schätzungsfehler der Markerpose ist stark vom Blickwinkel und ihrer Lage im Blickfeld abhängig [Rav15]. Bei einer großen perspetivischen Verzerrung ist die Schätzung dagegen genauer. Desweiteren schränkt die Auflösung auch die Distanz ein, bis zu welcher der Marker im Bild als solches erkannt wird.

- Mehrdeutigkeit der Pose:
  Bei bestimmten Konstellationen des Markers relativ zur Kamera ist dieser nicht eindeutig mit einem Kamerabild zuordenbar. Genauere Details können aus [TSM14] entnommen werden.

3.1.1 Posenfilterung

Um die AR Marker für die Lokalisierung zu verwenden, sollte der Messfehler möglichst konsistent und klein gehalten werden. Daher ist der hier verfolgte Ansatz die Filterung „invalider“ Posenschätzungen. Diese beinhalten Posen, die in perspektivischer Hinsicht ungünstig sind. D.h. in der Bildebene betrachtet wird gefordert, dass folgende Beziehungen gelten:

$$\min(\{d_{kante,i}, i \in \{1,2,3,4\}\}) >= \epsilon_{kante} \tag{3.1}$$

$$|d_{kante,1} - d_{kante,3}| + |d_{kante,2} - d_{kante,4}| >= \epsilon_{verzerrung} \tag{3.2}$$

Dabei ist $d_{kante,i}$ als die Länge[1] der i-ten Markerkante definiert. ϵ_{kante} und $\epsilon_{verzerrung}$ stellen die unteren Schranken für die Kantenlänge und Verzerrung dar. Zu beachten ist, dass die mathematische Definition der Bildverzerrung in Glg. 3.2 willkürlich ist und ggf. durch eine geeignetere Darstellung ersetzt werden kann.

3.1.2 Projektion der Markerpose auf die 2D-Ebene

Über die optischen Gleichungen wird der Marker im Koordinatensystem der Kamera erfasst. Auch die Lage der Kamera ist im Koordinatensystem bzgl. seiner Pose bekannt, da die Kamera an einem fixierten Punkt des Roboters befestigt ist. Dadurch lässt sich eine Messung der Kamera als

$$^{R}T_{M} = {}^{R}T_{S} \cdot {}^{S}T_{M} \tag{3.3}$$

beschreiben, wobei die Indizes R, S und M entsprechend für Roboter, Sensor und Marker stehen. Dabei ist die Transformation ${}^{R}T_{S}$ über die Konstruktion gegeben und ${}^{S}T_{M}$ die tatsächliche Posenmessung aus den Bilddaten.

Die Transformation aus der Glg. 3.3 stellt die Markerpose in dreidimensionaler Form dar. Aufgrund der in Kap. 3.2 geeinigten Konvention der Posendarstellung in zweidimensionaler Form, muss die Messung der Pose auf die x-y-Ebene (der Navigationsebene) des Roboters projeziert werden. Das Detektionsmodul[2] der AR-Marker definiert den Ursprung des Markers so, dass die Bildebene in der x-y-Ebene liegt und die z-Achse aus dieser Ebene herauszeigt (siehe [Doc18]). Um die Anzahl der Singularitäten im Anwendungsszenario klein zu halten, wird daher die Projektion der z-Achse für die Orientierungsrichtung der 2D-Pose verwendet. In diesem Fall sind die Markerausrichtungen, dessen Bildebene parallel zur Navigationsebene liegen, nicht eindeutig definiert.

Nach der Projektion sind die Markermessungen als relative zweidimensionale Pose bzgl. $\mathbf{x}_r$ gegeben, welche für den Kalman Filter (genaueres in Kap. 3.2) weiterverwendet werden. Im Gegensatz zu üblichen Landmarkenmessungen in der Literatur (z.B. [MSW02]) wird hier nicht auf das *range and bearing* Prinzip zurückgegriffen, welches Landmarken über Entfernung und Winkel zum Objekt erfasst. Dabei unterscheidet sich die Darstellung der Position in Polarkoordinaten oder in kartesischen Koordinaten im Hinblick auf den Informationsgehalt kaum. Die Besonderheit in der Messung ist, dass die Orientierung

[1] Da bei einer Eckpunkterkennung Pixelkoordinaten des Bildes zurückgegeben werden, ist die Einheit auch in Pixeln. Da die Umwandlung der Werte in SI-Einheiten über die Kameraparameter nur ein konstanter Skalierungsfaktor ist, wird diese ausgelassen.
[2] siehe Kap. 5

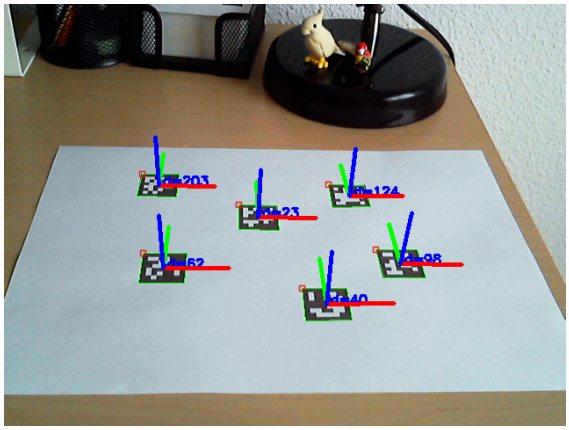
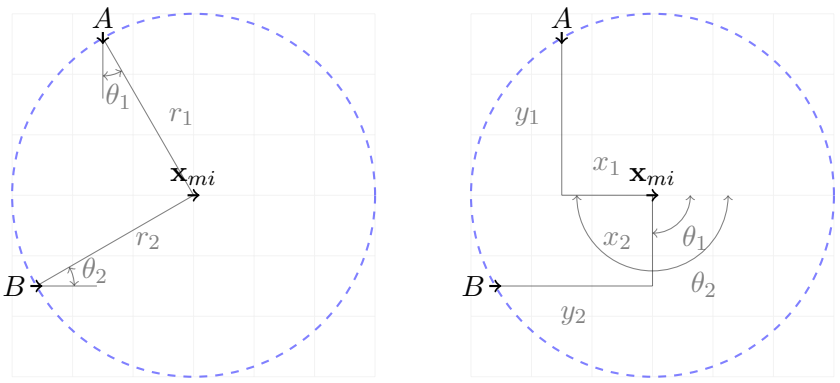


Abbildung 3.3: *Orientierung und Lage des Markerursprungs nach Detektion der Marker mit ArUco Bibliothek von OpenCV. Die roten, grünen und blauen Achsen stellen entsprechend in kartesischen Koordinaten die x-, y-, z-Achsen dar [Doc18].*

des erkannten Objekts eine zusätzliche Informationskomponente enthält, die für die Lokalisierung verwendet werden kann. Konkret wird das am Beispiel einer Messung erklärt, die in Abb. 3.4 visualisiert ist.



(a) *range and bearing Messung.* ***(b)*** *2D Posenmessung.*

Abbildung 3.4: *Vergleich von range and bearing Messung mit der 2D Posenmessung.*

Bei der beispielhaften Betrachtung der zwei Roboterposen A und B zu einem Marker $\mathbf{x}_{mi}$, bei denen der Abstand zum Marker und der beobachtete Winkel identisch sind, sind die Messungen voneinander nicht unterscheidbar. Es existieren auf dem Kreis um den Marker herum unendlich viele Roboterposen, die zu der gleichen Messung führen. Mithilfe der Orientierung des Markers im Raum kann zwischen den Posen auf der Kreislinie unterschieden werden.

Im Fall des Kalman-Filters, welcher in den folgenden Unterkapiteln 3.2.1 und 3.2.2 näher erläutert wird, wird im Korrekturschritt die Pose des Roboters im Fall von Abb. 3.4a

auf die Kreislinie hinoptimiert, während die Pose im Fall von Abb. 3.4b gegen die echte Pose konvergiert. Bei großen Schätzfehlern wird daher bei der Messung mit drei Freiheitsgraden eine schnellere Konvergenz als mit zwei freien Parametern erwartet. Eine nähere Untersuchung der Konvergenzeigenschaften wurde in dieser Arbeit nicht durchgeführt.

3.1.3 Probleme der AR-Markererkennung

Beim Anwendungsszenario der Indoor-Umgebung sind mögliche Gegenstände, mit denen der Roboter in Kontakt kommt, statische Objekte wie Wände, Säulen oder Decken. In großen Räumen kann aufgrund der limitierten Auflösung der Kameras nicht immer gewährleistet werden, dass „valide“ Marker im sichtbaren Bereich des Roboters liegen. Ein weiterer Fall, in der das Marker-basierte System suboptimal funktionieren würde, liegt vor, wenn die Sicht des Roboters auf die Marker verdeckt ist. Dabei reicht es aus die Sicht auf den Marker teilweise zu unterbrechen, um die Messung unbrauchbar zu machen.

3.2 Posenschätzung mit Hilfe des Kalman-Filters

Der Filter dient dazu Messdaten mit der Modellprädiktion zu verbinden, um dabei die Ungenauigkeiten im Modell und in der Messung zu berücksichtigen. Wie in Kap. 2.2.1 beschrieben, ist der Kalman-Filter ein Gaußscher Filter, der die Systemzustände als probabilistische Verteilungen beschreibt. Die Hauptannahme dabei ist, dass die Systemgrößen und Messgrößen in realen Systemen immer rauschbehaftet sind. Ist das modellierte System linear, dann schätzt der Kalman-Filter den Systemzustand optimal[3]. Ein Anfangszustand muss bekannt sein, da die Schätzung iterativ erfolgt. Daraus folgt, dass das Problem der globalen Zustandsschätzung nicht explizit adressiert wird, obwohl dies modellierbar ist (Gaußverteilung mit sehr hoher Varianz).

Allgemein kann das Zustandsraummodell folgendermaßen beschrieben werden

$$\begin{aligned} \mathbf{x}_{k+1} &= f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k \\ \mathbf{y}_k &= h(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{v}_k \end{aligned} \tag{3.4}$$

Die Funktionen f und h stellen entsprechend System- und Observationsmodell dar. Die dazugehörigen Rauschterme sind im Kalman-Filter mit einem Erwartungswert von Null normalverteilt, also $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$ und $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$. Die Unsicherheit des Zustands $\mathbf{x}_k$ wird mit der Kovarianzmatrix $\mathbf{P}_k$ notiert.

Im Zusammenhang des vorliegenden Systems sind die Unterfunktionen f_r und h_{mi}

[3] Optimierung des Signal-Rausch-Verhältnisses (SNR)

folgendermaßen definiert:

$$f_r(\mathbf{x}_k, \mathbf{u}_k) = \begin{pmatrix} x_{r,k} - \Delta d_k \cdot \sin(\theta_{r,k}) \\ y_{r,k} + \Delta d_k \cdot \cos(\theta_{r,k}) \\ \theta_{r,k} - \Delta\theta_k \end{pmatrix} \tag{3.5}$$

$$h_{mi}(\mathbf{x}_k, \mathbf{u}_k) = \begin{pmatrix} x_{mi,k}\cos(\theta_{r,k}) + y_{mi,k}\sin(\theta_{r,k}) - x_{r,k}\cos(\theta_{r,k}) - y_{r,k}\sin(\theta_{r,k}) \\ -x_{mi,k}\sin(\theta_{r,k}) + y_{mi,k}\cos(\theta_{r,k}) + x_{r,k}\sin(\theta_{r,k}) - y_{r,k}\cos(\theta_{r,k}) \\ \theta_m - \theta_{r,k} \end{pmatrix} \tag{3.6}$$

Der Index $k \in \mathbb{N}_0$ notiert den k-ten Zeitschritt auf der Zeitachse. Der allgemeine Zustandsvektor $\mathbf{x}_k$ setzt sich aus der Roboterkonfiguration $\mathbf{x}_{r,k} = (x_{r,k}, y_{r,k}, \theta_{r,k})^T$ und den dynamisch hinzugefügten Markerkonfigurationen $\mathbf{x}_{mi,k} = (x_{mi,k}, y_{mi,k}, \theta_{mi,k})^T$ mit Markerindex $i \in \mathbb{N}_0$ zusammen. Dabei wird die zweidimensionale Repräsentation der Posen angenommen. Wichtig dabei ist, dass die Markerpose $\mathbf{x}_{mi,k}$ im gleichen Referenzkoordinatensystem liegt wie $\mathbf{x}_{r,k}$. Anders als in Kap. 2.1 wird für den Eingang nicht der Vektor der Geschwindigkeiten $\mathbf{u}_{speed,k}$ verwendet, sondern die daraus abgeleiteten Odometriewerte $\mathbf{u}_k = (\Delta d_k, \Delta\theta_k)^T$, d.h.

$$\mathbf{u}_k = \int_{t_{start,k}}^{t_{end,k}} \mathbf{u}_{speed,k}\, dt \doteq \mathbf{u}_{speed,k}(t_{end,k} - t_{start,k}). \tag{3.7}$$

Das Systemmodell f_r bezieht sich nur auf die Roboterkonfiguration $\mathbf{x}_r$. f hingegen beschreibt die Systemdynamik des allgemeinen Zustands $\mathbf{x}$. Da die Markerposen sich nicht mit der Bewegung des Roboters verändern, werden die restlichen Einträge der mehrdimensionalen Funktion mit Nullfunktionen aufgefüllt, also $f = (f_r 0 \dots 0)^T$. Das Observationsmodell h hängt von der Messung $\mathbf{z}_k$ ab und setzt sich aus Funktionen mit der Form von Gleichung 3.6 zusammen.

3.2.1 Implementierung des Extended Kalman-Filter

Es ist zu erkennen, dass Glg. 3.5 und 3.6 nicht-linear sind. Da die Gleichungen des Kalman-Filters nur auf lineare Systeme anwendbar sind, ist es möglich das System in jeder Iteration durch das linearisierte System anzunähern. Das Resultat wird in der Literatur Extended Kalman Filter genannt ([S T05]).

Durch die Ableitung der Glg. 3.5 nach $\mathbf{x}_k$ und $\mathbf{u}_k$ werden die Jakobimatrizen $\mathbf{J}_{x_r}$ und

$\mathbf{J}_{u_r}$ bestimmt:

$$\mathbf{J}_{x_r} = \begin{pmatrix} 1 & 0 & -\Delta d \cos(\theta_r) \\ 0 & 1 & -\Delta d \sin(\theta_r) \\ 0 & 0 & 1 \end{pmatrix} \tag{3.8}$$

$$\mathbf{J}_{u_r} = \begin{pmatrix} -\sin(\theta_r) & 0 \\ -\cos(\theta_r) & 0 \\ 0 & 1 \end{pmatrix} \tag{3.9}$$

Die Jakobimatrix $\mathbf{H}$ für das Observationsmodell h setzt sich aus den Untermatrizen $\hat{\mathbf{H}}_{ri} = \frac{\partial h_{mi}}{\partial \mathbf{x}_r}$ und $\hat{\mathbf{H}}_{mi} = \frac{\partial h_{mi}}{\partial \mathbf{x}_{mi}}$ zusammen.

$$\hat{\mathbf{H}}_{ri} = \begin{pmatrix} -\cos(\theta_r) & -\sin(\theta_r) & -(x_{mi} - x_r)\sin(\theta_r) + (y_{mi} - y_r)\cos(\theta_r) \\ \sin(\theta_r) & -\cos(\theta_r) & -(x_{mi} - x_r)\cos(\theta_r) - (y_{mi} - y_r)\sin(\theta_r) \\ 0 & 0 & 1 \end{pmatrix} \tag{3.10}$$

$$\hat{\mathbf{H}}_{mi} = \begin{pmatrix} \cos(\theta_r) & \sin(\theta_r) & 0 \\ -\sin(\theta_r) & \cos(\theta_r) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.11}$$

Die Beobachtungsmatrix $\mathbf{H}$ ist je nach Messung $\mathbf{z}$ eine Kombination aus

$$\mathbf{H}_{mi} = \begin{pmatrix} \hat{\mathbf{H}}_{ri} & \mathbf{0} \dots \mathbf{0} & \hat{\mathbf{H}}_{mi} & \mathbf{0} \dots \mathbf{0} \end{pmatrix}. \tag{3.12}$$

Die Anordnung von $\hat{\mathbf{H}}_{mi}$ innerhalb von $\mathbf{H}_{mi}$ hängt von der Lage des i-ten Markers im Zustandsvektor ab. Ist die Reihenfolge sortiert, also $\mathbf{x}_k = (\mathbf{x}_{r,k}, \mathbf{x}_{m1}, \dots, \mathbf{x}_{mn})$, dann ist $\hat{\mathbf{H}}_{mi}$ der (i+1)-te Eintrag im Zeilenvektor mit den 3x3 Matrizen als Elemente.

Um eine Zusammensetzung der finalen Observationsmatrix $\mathbf{H}$ zu demonstrieren, wird zunächst einmal angenommen, dass bereits zwei Marker mit der ID 5 und 4 in dieser Reihenfolge in den Zustand aufgenommen wurden. Dann ergibt sich der Zustandsvektor

$$\mathbf{x} = (x_r, y_r, \theta_r, x_{m5}, y_{m5}, \theta_{m5}, x_{m4}, y_{m4}, \theta_{m4})^T. \tag{3.13}$$

Bei einer Observation[4] $\mathbf{z} = (\mathbf{z}_{m4}, \mathbf{z}_{m5})^T$ ergibt sich entsprechend

$$\mathbf{H} = \begin{pmatrix} \hat{\mathbf{H}}_{r4} & \mathbf{0} & \hat{\mathbf{H}}_{m4} \\ \hat{\mathbf{H}}_{r5} & \hat{\mathbf{H}}_{m5} & \mathbf{0} \end{pmatrix}. \tag{3.14}$$

Zu beachten ist, dass sich die Lage der Untermatrix $\hat{\mathbf{H}}_{mi}$ in den Spalten nach der

[4] Dabei notiert der Index entsprechend die Messung des entsprechenden Markers

Anordnung von $\mathbf{x}_{mi}$ im Zustandsvektor zeilenweise richtet. Die spaltenweise Anordnung von $\mathbf{H}_{mi}$ in der Observationsmatrix $\mathbf{H}$ richtet sich hingegen nach den Einträgen von $\mathbf{z}$.

Da die Matrizen für die Zustandsraumdarstellung bekannt sind, können die Übergangsgleichungen der Zustandsschätzung bestimmt werden. Diese folgen analog zu den hergeleiteten Formeln des linearen Kalman-Filters (siehe [S T05]). Der Unterschied hierbei ist, dass nun die Jakobimatrizen als Annäherung für das lineare System dienen und abhängig vom Arbeitspunkt sind (siehe Glg. 3.8, 3.9, 3.10 und 3.11). Die Prädiktions- und Korrekturschritte werden dann nach Gleichung 3.15 und 3.16 berechnet:

$$\begin{aligned} \mathbf{x}_{k|k-1} &= f(\mathbf{x}_{k-1|k-1}, \mathbf{u}_k) \\ \mathbf{P}_{k|k-1} &= \mathbf{J}_{x_r}\mathbf{P}_{k-1|k-1}\mathbf{J}_{x_r}^T + \mathbf{J}_{u_r}\mathbf{Q}\mathbf{J}_{u_r}^T \end{aligned} \tag{3.15}$$

$$\begin{aligned} \tilde{\mathbf{y}}_k &= \mathbf{z}_k - h(\mathbf{x}_{k|k-1}, \mathbf{u}_k) \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k) \\ \mathbf{x}_{k|k} &= \mathbf{x}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \end{aligned} \tag{3.16}$$

Um zwischen Prädiktion und Korrektur zu unterscheiden, ist der Zeitindex zweiteilig. In der Praxis ist die Abfolge der Odometrieeingänge und Messungen nicht zwangsweise synchron. D.h. es können bei einer fehlenden Messung mehrere Prädiktionsschritte bzw. bei mehreren Sensoren auch einige Korrekturschritte aufeinanderfolgen. Demnach werden Glg. 3.15 und 3.16 als unabhängige Zustandspropagationen bzw. Kovarianzpropagationen betrachtet, welche auch entsprechend implementiert wurden (siehe Kap. 5).

3.2.2 Implementierung des Unscented Kalman-Filter

Die Nichtlinearität des Systems wird vom EKF in Kap. 3.2.1 über die Linearisierung am Arbeitspunkt aufgelöst, wodurch für die Berechnung der Kovarianzmatrix $\mathbf{P}$ die Formeln des linearen Filters anwendbar sind. Ein alternativer Ansatz zum EKF wurde in der Arbeit von Wan et al. ([WM00]) auf Basis des *Unscented Transform* vorgestellt.

Das Grundprinzip ist die Darstellung der Kovarianz des Systems mithilfe charakteristischer Punkte (sigma points), die durch das nicht-lineare System propagieren. Nach Wan et al. erzielt diese Methode eine Approximationsgenauigkeit bis zum Taylorpolynom dritter Ordnung. Im Gegensatz zum EKF, welches die nicht-linearen Funktionen über das Taylorpolynom ersten Grades (Jakobimatrix) annähern, verspricht diese Methode eine höhere Genauigkeit.

Algorithmus 3.1: *Prädiktionsschritt des UKF [WM00].*

algorithm ukf_predict($\mathbf{x}_{k|k-1}$, $\mathbf{P}_{k|k-1}$, u_k)
input: state mean $\mathbf{x}_{k-1|k-1}$, state covariance $\mathbf{P}_{k-1|k-1}$, odometry u_k
output: state mean $\mathbf{x}_{k|k-1}$, state covariance $\mathbf{P}_{k|k-1}$
begin
  $\chi^{(i)} \leftarrow$ generate sigma points

  // calculate model prediction
  foreach: $\chi^{(i)}$
    $\hat{\chi}^{(i)} \leftarrow f(\chi^{(i)}, u_k)$
  end

  // update distribution on prediction
  $\mathbf{x}_{k|k-1} \leftarrow \sum w_m^{(i)} \hat{\chi}^{(i)}$
  $\mathbf{P}_{k|k-1} \leftarrow \sum w_c^{(i)} (\hat{\chi}^{(i)} - \mathbf{x}_{k|k-1})(\hat{\chi}^{(i)} - \mathbf{x}_{k|k-1})^T + \mathbf{Q}_k$

  return $\mathbf{x}_{k|k-1}$, $\mathbf{P}_{k|k-1}$
end

Algorithmus 3.2: *Korrekturschritt des UKF [WM00].*

algorithm ukf_correct($\mathbf{x}_{k|k-1}$, $\mathbf{P}_{k|k-1}$, y_k)
input: state mean $\mathbf{x}_{k|k-1}$, state covariance $\mathbf{P}_{k|k-1}$, measurement y_k
output: state mean $\mathbf{x}_{k|k}$, state covariance $\mathbf{P}_{k|k}$
begin
  $\chi^{(i)} \leftarrow$ generate sigma points

  // calculate measurement prediction distribution
  foreach: $\chi^{(i)}$
    $\hat{y}^{(i)} \leftarrow h(\chi^{(i)})$
  end

  // calculate measurement prediction mean, auto- and crosscorrelation
  $\hat{y}_k \leftarrow \sum w_m^{(i)} \hat{y}^{(i)}$
  $\mathbf{P}_{yy} \leftarrow \sum w_c^{(i)} (\hat{y}^{(i)} - \hat{y}_k)(\hat{y}^{(i)} - \hat{y}_k)^T + \mathbf{R}_k$
  $\mathbf{P}_{xy} \leftarrow \sum w_c^{(i)} (\chi^{(i)} - \mathbf{x}_{k|k-1})(\hat{y}^{(i)} - \hat{y}_k)^T$

  // update distribution on prediction
  $\mathbf{K}_k = \mathbf{P}_{xy}\mathbf{P}_{yy}^{-1}$
  $\mathbf{x}_{k|k} \leftarrow \mathbf{x}_{k|k-1} + \mathbf{K}_k(y_k - \hat{y}_k)$
  $\mathbf{P}_{k|k} \leftarrow \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{P}_{yy}\mathbf{K}_k^T$

  return $\mathbf{x}_{k|k}$, $\mathbf{P}_{k|k}$
end

In Alg. 3.1 und 3.2 wird das Funktionsprinzip des UKF skizziert. Die Gewichtungen (genaueres in Anhang A.1) $w_c^{(i)}$ und $w_m^{(i)}$, die für die Rücktranformation der charakteristischen Punkte in Mittelwert und Kovarianz verantwortlich sind, können vorkalkuliert werden, um Rechenzeit zu sparen. Es ist zu erkennen, dass auch hier die zwischenzeitige Repräsentation der Unsicherheit als Gaußglocke bestehen bleibt. Dennoch werden andere

Verteilungen über die Sigmapunkte berücksichtigt.

In Analogie zum EKF ist beim UKF der Prädiktionsschritt und der Korrekturschritt voneinander unabhängig. Somit ist auch hier eine seperate Implementierung möglich. Theoretisch betrachtet, ist dadurch eine Kombination aus beiden Filtermethoden anwendbar. Beispielsweise kann der Prädiktionsschritt vom UKF übernommen werden, während der Korrekturschritt vom EKF ausgeführt wird. Dies wurde in dieser Arbeit nicht untersucht, sollte aber aufgrund der unterschiedlichen Eigenschaften der Filter für zukünftige Untersuchungen in Betracht gezogen werden.

3.2.3 Initialisierung dynamisch erkannter Marker

Die komplexe Anordnung der Untermatrizen entsprechend der Markerindizes aus Kap. 3.2.1 hängt vor allem von der Initialisierung der Marker im Zustand ab. Da hier nicht alle Indizes sortiert im Zustand vorliegen, ist die kompakte Form möglich. Natürlich ist die Anordnung wie die der *sparse matrix* vom Implementierungsaufwand einfacher, hat aber schwerwiegende Nachteile in der Speicherkomplexität. Vor allem bei möglichen 1024 verschiedenen Markern skaliert sich der Unterschied zum dynamisch angelegtem Zustand schnell, wenn nur wenige davon verwendet werden.

Diese Implementierung benötigt entsprechend einen Initialisierungsmechanismus, der in einer Toolbox von P. Corke ([Cor96]) verwendet wurde und im Folgenden vorgestellt wird. Bei Markerobservationen, dessen ID nicht im Zustandsvektor vorhanden ist, wird eine Subroutine eingeleitet. Dazu wird der vorhandene Zustandsvektor, um den zusätzlichen Vektor

$$\mathbf{x}_{mi} = \begin{pmatrix} x_r + x_{mi,mess}\cos(\theta_r) - y_{mi,mess}\sin(\theta_r) \\ y_r + x_{mi,mess}\sin(\theta_r) + y_{mi,mess}\cos(\theta_r) \\ \theta_r + \theta_{mi,mess} \end{pmatrix} \tag{3.17}$$

erweitert, sodass $\mathbf{x} \rightarrow (\mathbf{x}, \mathbf{x}_{mi})^T$.

Die Unsicherheit der Markermessung muss ebenfalls in der Kovarianzmatrix $\mathbf{P}$ berücksichtigt werden. Die Jakobimatrix von Glg. 3.17

$$\mathbf{L} = \begin{pmatrix} \cos(\theta_r) & -\sin(\theta_r) & 0 \\ \sin(\theta_r) & \cos(\theta_r) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.18}$$

transformiert das Messrauschen $\mathbf{R}$ in den Zustandsraum, sodass schließlich mit

$$\mathbf{P} \rightarrow \begin{pmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{LRL}^T \end{pmatrix} \tag{3.19}$$

die neue Kovarianzmatrix folgt.

3.2.4 Synchronisationsmechanismus der asynchron eintreffenden Sensordaten

Die Umsetzung der in den vorherigen Kapiteln vorgestellten Verfahren ist einfach, wenn die sensorischen Informationen in synchroner Reihenfolge vorliegen und abgearbeitet werden können. Dies ist in der Realität meistens nicht der Fall. So hat zum Beispiel eine Kamera eine Bildfrequenz von 30 Hz, aber die Odemetrie liefert hingegen ihre Werte mit einer Frequenz von beispielsweise 20 Hz. Dadurch folgen unweigerlich mehrere Kameramessungen aufeinander. Außerdem kann bei einem aufgenommenen Bild kein entdeckter Marker sein, sodass diese Messung hinfällig und für den Filter nicht verwertbar ist. Dadurch kann der Fall auftreten, dass für einen bestimmten Zeitraum keine Kameramessungen vorliegen und der Roboter nur auf Odometriedaten fährt. Abb. 3.5 zeigt ein abstraktes Beispiel ankommender Datenpakete in Echtzeit.

Odom	Cam1	Odom	Cam2	Cam1	Cam2	Cam1	Cam3	Odom	Odom

Abbildung 3.5: *Asynchron ankommende Datenpakete für die Posenschätzung.*

Deswegen wurde für diese Arbeit ein einfacher Sychronisationsmechanismus implementiert. Dieser geht davon aus, dass Odometriedaten für die Bewegung des Roboters notwendig sind. Dadurch muss für jeden Zyklus ein Prädiktionsschritt vorhanden sein. Auf diesen Schritt kann maximal ein Korrekturschritt pro Sensor erfolgen. Bei einem Kamerasystem aus vier Kameras wird jeweils die erste Messung mit mindestens einem Marker für den Korrekturschritt verwendet, sodass maximal vier Korrekturschritte auf einer Prädiktion folgen können.

Odom	Cam1	Odom	Cam2	Cam1	Cam3	Odom	Odom

Abbildung 3.6: *Synchronisierte Reihenfolge der Datenpakete des Beispiels aus Abb. 3.5.*

Diese Methode vernachlässigt somit alle Bildinformationen, die in dem Zeitraum zwischen erstem Korrekturschritt und dem nächstem Prädiktionsschritt gesammelt werden. Im Beispiel von Abb. 3.5 fallen dadurch redundante Datenpakete nach dem Odometrieschritt weg und daraus resultiert Abb. 3.6. Bei kleinen Bewegungen sind die Bilder aufgrund der Einschränkungen in der Auflösung nahezu identisch und liefern für unsere Lokalisierung keine neuen Informationen.

3.2.5 Verwaltung der statisch und dynamisch angelegten Markerkarte

In Kap. 3.2.3 wurden die Posen der Marker im Zustand angelegt und mit dem Kalman-Filter mitgeschätzt. Dabei wird davon ausgegangen, dass über diese Marker keine Informationen bekannt sind. In dem Fall, dass die feste Position und Orientierung vorher bekannt sind, ist ihre Schätzung überflüssig. Aus diesem Grund wurde hier ein Entscheidungsmechanismus verwendet, welcher in Abb. 3.7 skizziert wird.

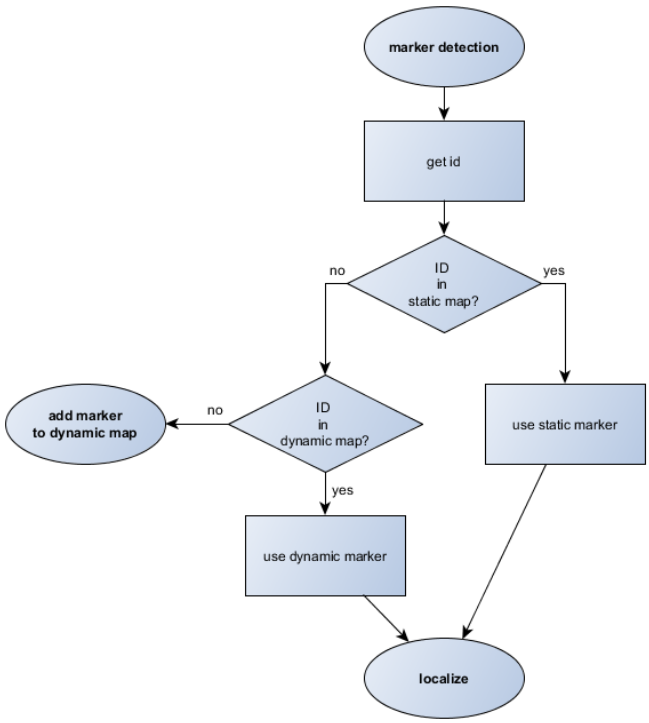


Abbildung 3.7: *Markerverwaltung*

Die statische Karte kann über verschiedene Mechanismen eingelesen werden und liegt dem Algorithmus als *lookup table* vor. Die Verwendung der statischen und dynamischen Karte im Kalman-Filter ist von der Implementierung her gleich, da die Werte von $\mathbf{x}_{mi}$ als solches vorliegen (im statischen Fall als exakter Wert und im dynamischen Fall als Mittelwert der Verteilung).

3.2.6 Vor- und Nachteile der verwendeten Verfahren

Es wurde in Kap. 2.2.1 darauf hingewiesen, dass in der Arbeit von Konatowski et al. ([Sta16]) der Qualitätsvergleich zwischen den verschiedenen Filterverfahren zugunsten des Partikelfilters ausgefallen ist. Dennoch wurde in dieser Arbeit sich für die Implemen-

tierung des EKFs und des UKFs entschieden. Dies hat folgende Gründe:

- **Performance**:
  Es wurde von Konatowski et al. erwähnt, dass eine genaue Darstellung der Zustandsverteilung mit dem Partikelfilter eine hohe Anzahl an Partikeln benötigt. Dabei wurde die Skalierung der Rechenintensität vernachlässigt. Da jeder Partikel eine eigene Repräsentation des Zustands besitzt, müssen die Berechnungen der Filterschritte sowie die der Gewichtung und die des Samplings für jeden einzelnen Partikel durchgeführt werden. Für eine reine Lokalisierung (in unserem Fall entsprechend drei Parametern im Zustand) ist die Auswirkung gering. Durch die Einbeziehung der dynamischen Featurekarte in den Zustand sowie die spätere Berücksichtigung der metrischen Rasterkarte, nimmt sowohl die Rechenkomplexität als auch die Speicherkomplexität quadratisch[5] zu. Im Vergleich dazu skaliert sich der Rechen- und Speicheraufwand beim Kalman-Filter mit der Anzahl an dynamisch hinzugefügten Markern. Es existiert dabei nur eine metrische Karte. Beim Partikelfilter jedoch würde jedes Partikel eine Rasterkarte besitzen.

- **Kompakte Darstellung der Wahrscheinlichkeitsverteilung**:
  Die Zustandsverteilung ist mit zwei Paramtern (Mittelwert und Kovarianz) eindeutig beschrieben. Beim Partikelfilter liegt diese indirekt über die Partikeldichte und Gewichtung vor. Bei der weiteren Verwendung der Schätzung in anderen Modulen hat die kompakte Darstellung Vorteile für ihre Weiterverarbeitung, dadurch dass die Konvertierung der Partikel in eine „beste“ Schätzung ausfällt.

Beim Vergleichen dieser Marker-basierten Lokalisierung mit anderen Feature-basierten Verfahren (z.B. [MSW02]), weist diese Methode dem Vorteil auf, dass das *Zuordnungsproblem* durch die eindeutige Definition der Markeridentität nicht vorhanden ist. Dennoch wird sich hier auf die Lokalisierung mit Hilfe der AR-Marker beschränkt und verliert dadurch Flexibilität.

Ein wichtiger Nachteil der Kalman-Filter-basierten Verfahren ist, dass die Schätzung des Zustands anfällig gegenüber Fehlmessungen ist. D.h. wenn der Korrekturschritt mit einer Fehlmessung[6] durchgeführt wird, dann hat diese Messung einen gleichen Stellenwert wie korrekte Messungen und wirkt sich somit entsprechend negativ auf die Schätzung aus. Da bei der AR-Markerdetektion häufig solche Fehlmessungen über gleichartige Objekte (z.B. Bildschirme, Aufkleber etc.) entstehen können, ist die Vorfilterung der Messungen für die Lokalisierungsqualität entscheidend.

[5] Linear mit der Anzahl dynamisch hinzugefügter Marker und linear mit der Anzahl Partikel $\mathcal{O}(mn)$
[6] Die Fehlmessung bezieht sich hier auf eine Messung, die nicht mit dem Messrauschung prädiziert werden (*anomaly*).

3.3 Rasterkartenerstellung bei bekannter Pose

Der SLAM Algorithmus setzt sich aus Lokalisierung und Kartierung zusammen. Der Teil der Kartierung bezeichnet in diesem Fall die Integration (Kap. 3.2.3) und das Tracking (Kap. 3.2.1 und 3.2.2) dynamischer Landmarken (**topologische Karte**). Diese Landmarken liefern Umgebungsinformationen für die Lokalisierung des Roboters, jedoch keine metrische Karte für die Navigation. Bei einem rein reaktivem System ist dies nicht zwingend notwendig (wie z.B. [FBT96]).

3.3.1 Motivation einer Rasterkarte

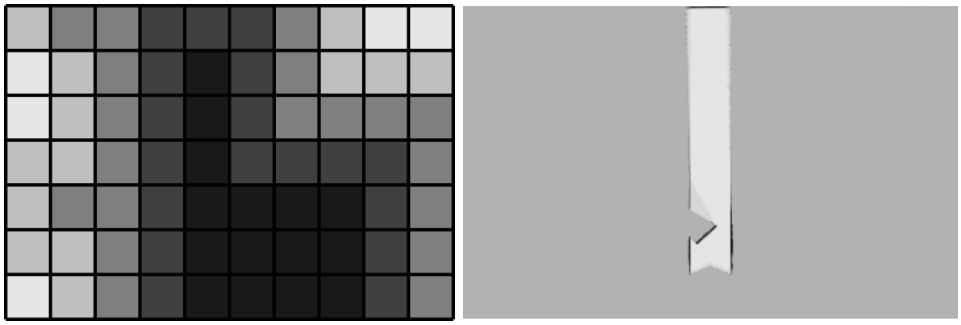
Die Implementierung einer zusätzlichen **Rasterkarte** ist aufgrund folgender Punkte sinnvoll:

- Pfadplanung
- Abdeckung nicht sichtbarer Bereiche zur Kollisionsvermeidung

Der erste Punkt bezieht sich dabei auf hybride Pfadplanungsmethoden, die überlagernd zu einem schnellen reaktiven Kollisionsvermeidungssystem eine globale Planung implementieren. Diese beheben die Nachteile einer lokalen Lösung. Um eine globale Planung durchführen zu können, muss eine Erfassung und Repräsentation des Navigationsraumes vorhanden sein. Der zweite Punkt bezieht sich speziell auf den Seekur Jr. Aufgrund des Aufbaus besitzt der mobile Roboter einen einzelnen Lasersensor auf der Vorderseite der Fahrplattform und deckt dementsprechend nur diesen Bereich ab (siehe Abb. 4.2a). Somit werden mögliche Bewegungsoperationen in Richtung der Seitenbereiche und der Rückseite nicht aktiv abgedeckt und werden in einer Momentanmessung der Sensorik nicht von der Kollisionsvermeidung mitberücksichtigt. In diesen Bereichen können Kollisionen mit dem Seekur Jr auftreten. Aus diesem Grund bietet sich die Rasterkarte an, welche die Historie der Laserscannerpunkte inhärent speichert. Diese Punkte können im Nachhinein für die Kollisionsvermeidung berücksichtigt werden. Dazu wird detaillierter in Kap. 4 eingegangen.

3.3.2 Struktur der Rasterkarte

Genau wie bei der Pose des Roboters kann der Zustand der Umgebung auch im Fall der Rasterkarte probabilistisch betrachtet werden. D.h. in den Zellen der Karte ist die Besetzung nicht absolut, also entweder belegt oder frei, sondern besitzt eine Wahrscheinlichkeit, mit der sie belegt ist. Diese können mit dem Bayesfilter aktualisiert werden, sodass die Kartenänderungen allein auf den Messwerten beruhen.



(a) Rasterkarte in Nahansicht. *(b) Kartierung eines Flurs.*

Abbildung 3.8: *Probabilistische Rasterkarte mit Grauwerten zur Darstellung der Belegtheitswahrscheinlichkeit (weiß: unbelegt, schwarz: belegt).*

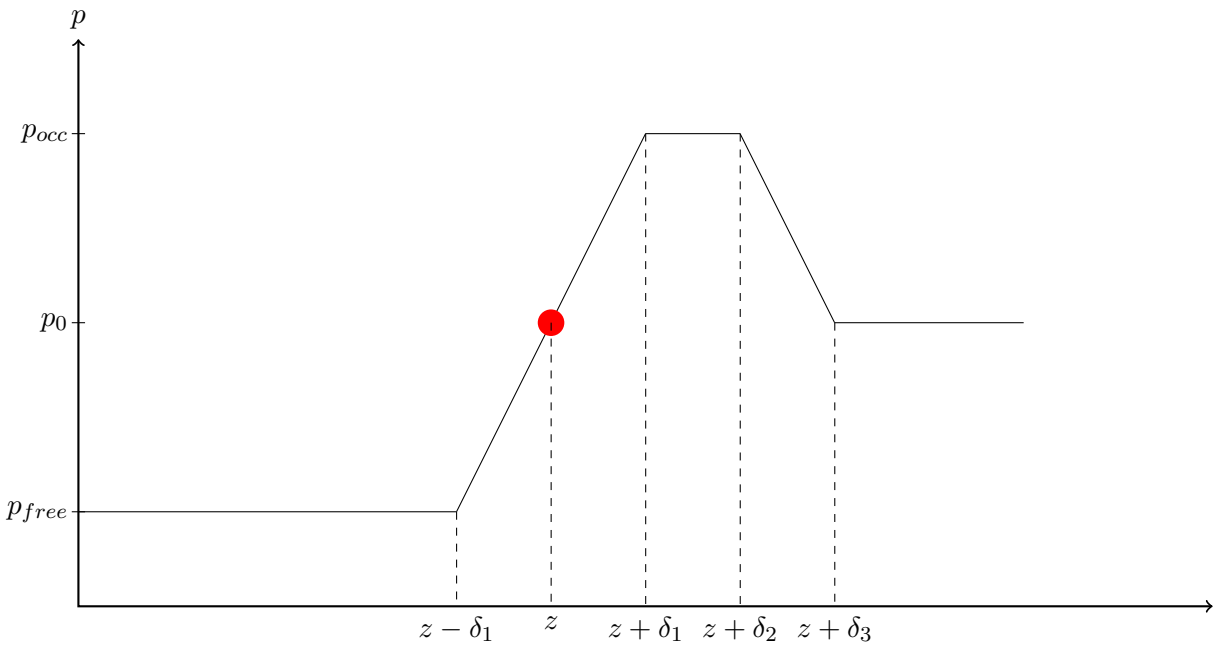


Abbildung 3.9: *Inverses Sensormodell des Laserscanners in 1D.*

Das hier verwendete Prinzip der Kartenaktualisierung basiert auf dem gleichen Algorithmus wie auch bei Moravec ([Mor89]). Zunächst wird ein **inverses Sensormodell** aufgestellt, welches die Wahrscheinlichkeitsverteilung $p(\mathbf{m}_{t,i}|\mathbf{z}_t, \mathbf{x}_t)$ eines Scans modelliert. Also wie wahrscheinlich es ist, dass die Kartenzellen $\mathbf{m}_{t,i}$ belegt sind mit der Bedingung, dass die Beobachtung $\mathbf{z}_t$ eingetroffen ist und der momentane Zustand (hier im speziellen die Roboterpose) $\mathbf{x}_t$ ist.

Abb. 3.9 zeigt das verwendete Sensormodell einer Längenmessung z im eindimensionalem Raum. Es ist zu erkennen, dass direkt hinter der Messung die Wahrscheinlichkeit,

dass ein Objekt in den Zellen vorhanden ist, am höchsten ist. Vor der Messung ist die Wahrscheinlichkeit entsprechend niedrig. Für die Werte weit hinter z (abhängig von den Parametern $\delta_1, \delta_2, \delta_3$) ist keine Information vorhanden und somit bleibt die a-priori Wahrscheinlichkeit bestehen bzw. wird nicht aktualisiert. Die Werte p_{occ} und p_{free} können in $(0, 1)$ frei gewählt werden. Dennoch beeinflussen sie die Konvergenzeigenschaften der Zellwahrscheinlichkeit.

3.3.3 Update der Rasterkarte durch eine Sensormessung

Die Zellen werden nach Algorithmus 3.3 bei einer Messung aktualisiert.

Algorithmus 3.3: *Updatemechanismus der Rasterkarte [S T05]*

```
1  algorithm occupancy_grid_mapping({l_{t-1,i}}, x_t, z_t)
2  input: log-odds {l_{t-1,i}}, robot state x_t, robot measurement z_t
3  output: log-odds {l_{t,i}}
4  begin
5    foreach m_i do
6      if m_i in perceptual field of z_t then
7        l_{t,i} = l_{t-1,i} + inverse_sensor_model(m_i, x_t, z_t) - l_0
8      else
9        l_{t,i} = l_{t-1,i}
10     endif
11   endfor
12   return {l_{t,i}}
13 end
```

Dabei bezeichnet l die *log-odds* Darstellung von der Belegungswahrscheinlichkeit p, um numerische Instabilitäten bei $p \to 0$ oder $p \to 1$ zu vermeiden. Ihre Definition lautet

$$l_{t,i} = \log \frac{p(\mathbf{m}_i|\mathbf{z}_t, \mathbf{x}_t)}{1 - p(\mathbf{m}_i|\mathbf{z}_t, \mathbf{x}_t)} \tag{3.20}$$

bzw. ihre Inverse

$$p(\mathbf{m}_i|\mathbf{z}_t, \mathbf{x}_t) = 1 - \frac{1}{1 + \exp(l_{t,i})} \tag{3.21}$$

Bei genauerer Betrachtung von Alg. 3.3 ist zu erkennen, dass die Problematik der Umwandlung des inversen Sensormodells in der eindimensionalen Distanzmessung in den zweidimensionalen Zustandsraum der Karte nicht näher betrachtet wird und mit der Unterfunktion **inverse_sensor_model**$(\mathbf{m}_i, x_t, z_t)$ abstrahiert wird. Da eine eindeutige Zuordnung einer Distanzmessung zu einer Zelle mit Hilfe von Rasteralgorithmen wie den Bresenhamalgorithmus im Allgemeinen nicht möglich ist, muss entschieden werden wie das Modell aus Abb. 3.9 angewendet werden kann. Die Literatur schlägt vor die Distanzmessung mit der geringsten Winkelabweichung[7] zur Zelle als Update zu verwenden

[7] Die Distanzmessungen des Laserscanners enthalten neben der Distanz ebenfalls den Winkel relativ zur

([S T05]). Dies wurde hier implementiert.

Neben dem Grundalgorithmus werden hier die maximale und minimale Zellbelegungswahrscheinlichkeit begrenzt. Dies hat zur Folge, dass Zellen, die über längere Zeit als frei gekennzeichnet wurden, schnell wieder in den belegten Zustand wechseln können. Dadurch können semi-statische Objekte in der Karte berücksichtigt werden.

Aus den probabilistischen Werten wird für die Navigation die tatsächlichen Belegung extrahiert. Dies wird durch ein einfaches *Thresholding* erreicht. D.h. alle Werte über einen gewissen Schwellwert p_{thresh} werden als belegt markiert, während die darunter als frei markiert werden, also

$$\begin{aligned} \mathbf{m}_i \text{ is occupied} &\leftarrow p(\mathbf{m}_i|\mathbf{z}_t, \mathbf{x}_t) \geq p_{thresh} \\ \mathbf{m}_i \text{ is free} &\leftarrow p(\mathbf{m}_i|\mathbf{z}_t, \mathbf{x}_t) < p_{thresh} \end{aligned} \tag{3.22}$$

Das Resultat des Thresholding ist dann eine Karte mit binären Werten, demnach eine *binary occupancy grid*. Diese kann für Pfadplanungen oder Kostenfunktionen manipuliert werden (siehe Kap. 4.3).

Sensorpose (*range and bearing*).

KAPITEL 4

Navigation

Wie schon in Kap. 1.2 erwähnt, ist das Ziel die Navigation über einen reaktiven kollisionsvermeidenden Ansatz zu realisieren. Der Fokus liegt dabei in der Sicherheit der Navigation. Demnach wird nicht das volle Spektrum der Bewegungsmöglichkeiten des Roboters ausgenutzt, sondern nur diese, die in ihrer Bewegung abgesichert werden können. Eine Veranschaulichung der Navigationsebenen ist in Abb. 4.1 abgebildet.

Die reaktive Navigation ist in zwei Teile aufgeteilt: zum einen in das Kollisionsvermeidungsmodul und zum anderen in die Sicherheitsschicht. Ersteres generiert auf Basis momentaner Sensor- und Karteninformationen eine für den Zeitpunkt optimale[1] Bewegung. Die Sicherheitsschicht hingegen überprüft, ob ein Zustand eingetroffen ist, bei der sich der Roboter nicht mehr weiterbewegen darf. In diesem Fall wird der Befehl der überliegenden Schicht ignoriert und es wird versucht den Roboter schnellstmöglich abzubremsen. Bei der momentanen Implementierung wird dazu einfach überprüft, ob sich Objekte in der unmittelbaren Umgebung befinden. Natürlich können in weiteren Arbeiten Prädiktionsmechanismen für dynamische Objekte mit in den Sicherheitsfilter integriert werden.

4.1 Nebenbedingungen des Seekur Jrs für die Navigation

Der Aufbau vom Seekur Jr kann neben seinem Bewegungsmodell (siehe Kap. 2.1) einen wichtigen Einfluss auf die Navigation haben. Dabei werden folgende Fragen aufgeworfen:

- Welcher Teil vom Roboter ist sensorisch mit Distanzmessern abgedeckt?
- Was ist der Umriss vom Roboter und welche Bewegungen ermöglicht/verhindert diese?

[1] Optimal in der Hinsicht vordefinierten Metrik

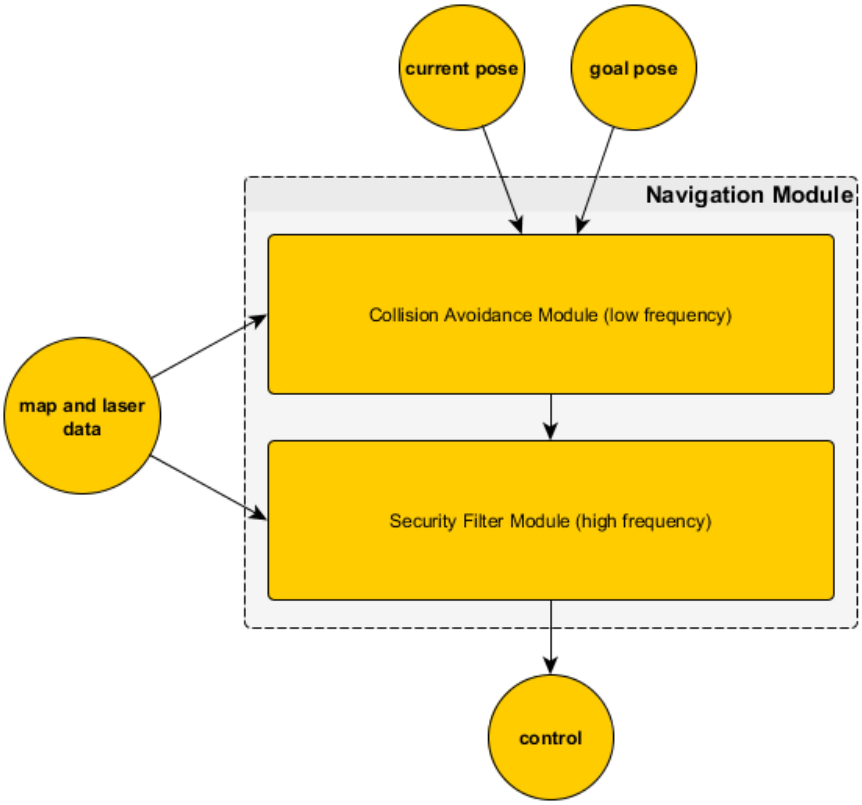
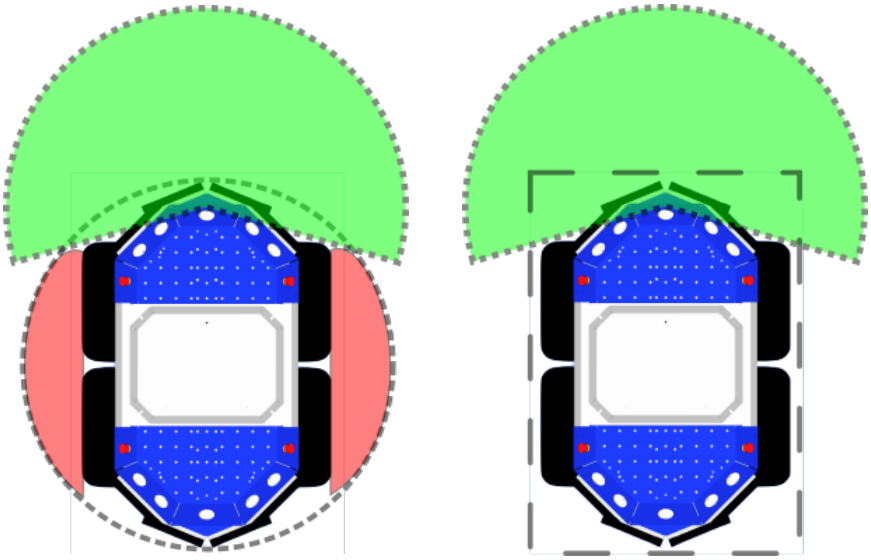


Abbildung 4.1: *Schichtenaufbau des Navigationsmoduls mit einem niederfrequentigen lokalen Planer für den Sollwert der Motoren und einem hochfrequentigen Filtermodul für Notabschaltungen.*

Im verwendeten Aufbau ist der Laserscanner des Seekur Jrs lediglich zur Vorderseite hin ausgerichtet. Deswegen sind nur Eingänge $\mathbf{u}_{speed}$ mit $u_v \geq 0$ zugelassen. Aufgrund der Bewegunskinematik des unicycle models bewegt sich der Roboter mit dieser Einschränkung tendenziell in den Bereich hinein, der vom Lasersensor erfasst wurde. Dennoch sind dadurch nicht alle Bewegungsbereiche ausgeschlossen, die nicht vom Sensor abgedeckt sind. Betrachtet man Abb. 4.2a, dann sind genau die roten Bereiche kritisch für die Kollisionsvermeidung. Objekte in diesem Gebiet können gerade bei reinen Rotationen nicht erfasst werden.

Die zweite Frage bezieht sich speziell auf die Bewegungsplanung. Der Umriss des Roboters entscheidet über die Komplexität der Planung und ihre Genauigkeit. Nehmen wir als Beispiel den Seekur Jr: In weiten Gassen und Räumen ist es irrelevant wie der Umriss des Seekur Jrs modelliert wird, solange der Roboter genug Abstand zu Objekten hält. In engen Gassen hingegen wie z.B. Türen wird der Unterschied bemerkbar. Da



(a) Kreisförmiges Gebiet als Referenz *(b) Rechteckiges Gebiet als Referenz*

Abbildung 4.2: *Umriss des SeekurJrs. In grün dargestellt ist der vom Sensor abgedeckte Bereich und in rot ist der tote Winkel des Roboters, der bei einer reinen Rotation betreten werden kann.*

der kreisförmige Umriss aus Abb. 4.2a eine Ausdehnung zur Seite hat, „passt“ der Roboter nicht mehr durch die Durchgänge, während der rechteckige Umriss aus 4.2b mit seiner schmalen Ausdehnung dadurch manövrieren kann. Bei den Arbeiten von Fox et al. ([Die97]) oder Maroti et al. ([MSK+13]) wurden aufgrund der einfachen Abfrageeigenschaften kreisförmige Umrisse verwendet. So kann man zum Beispiel aus einer Rasterkarte mit den kartierten Objekten direkt den Navigationsraum generieren, d.h. die Positionen, in dem der Roboter navigieren darf ohne mit Objekten zu kollidieren. Da dieser Umriss rotationsinvariant ist, wird nicht zwischen verschiedenen Posen mit der gleichen Position unterschieden. Dies schränkt den Suchraum für die Pfadplanung enorm ein. Dennoch ist diese Annäherung nicht für alle Roboter geeignet wie auch im Fall vom Seekur Jr. Andere Arbeiten erweitern die Kollisionsvermeidung mit alternativen *footprints* ([APT+02] oder [MM09]). Diese müssen nicht gezwungenermaßen einfach zusammenhängende Gebiete sein. Dadurch sind komplexe Bewegungsplanungen mit mehreren Gebieten möglich. Dies wäre besonders für mobile Roboter mit zusätzlichen Aktoren interessant, wie z.B. der Seekur Jr mit montiertem Roboterarm oder verbundenen Arbeiterdrohnen.

In dieser Arbeit wird der Roboterumriss als rechteckig betrachtet. Wie auch bei der Arbeit von Arras et al. ([APT+02]) beschränkt sich die Kollisionsabfrage im Roboterkoordinatensystem auf geometrische Operationen. Unter Anwendung des in Kap. 4.2 vorgestellten Algorithmus DWA hat dies nicht nur Vorteile hinsichtlich engen Passagen,

sondern auch Auswirkungen auf die Trajektorie, welche in Kap. 6.2.1 näher erläutert werden.

4.2 Kollisionsvermeidung

Der hier verwendete Algorithmus nennt sich Dynamic Window Approach. Wie auch bei Lokalisierung in Kap. 3.2 wird sich bei der Navigation auf die zweidimensionale Plane beschränkt, somit ist der Navigationsraum ebenfalls zweidimensional. Der DWA beruht auf dem Bewegungsmodell in Kap. 2.1, die stückweise konstante Geschwindigkeitsphasen zur Modellierung der Trajektorie verwendet. Fox et al. haben gezeigt, dass der Approximationsfehler der Trajektorie beschränkt ist und somit als Näherung verwendet werden kann ([Die97]).

4.2.1 Framework der Kollsionsvermeidung basierend auf dem Dynamic Window Approach

Der DWA wurde erstmals in einer Arbeit von Fox et al. ([FBT96]) vorgestellt und baut darauf auf, dass der Geschwindigkeitvektor $\mathbf{u}_{speed} = (u_v, u_\omega)^T$ als Eingang für die Posenbewegung des Roboters dient. Da das verwendete *unicycle model* den Geschwindigkeitsvektor als Systemeingang verwendet, bietet sich dies in der vorliegenden Arbeit ebenfalls an. Bei anderen z.B. auf Odemetrie basierenden Modellen kann einfach in den Geschwindigkeitsraum transformiert werden. Bei bekannter momentaner Geschwindigkeit wird aus einem Bereich um dieser herum (*Dynamic Window*) Samples abgetastet. Der Sinn dahinter ist, dass die Dynamik des Navigationssystems nur diese in einem Zeitschritt zulässt. Gründe dafür sind z.B. begrenztes Drehmoment der Motoren oder generelle Trägheit. Die abgetasteten Geschwindigkeitsvektoren unterliegen dabei

- hard constraints:
  Mit den sogenannten *hard constraints* werden die Geschwindigkeitspaare $\mathbf{u}_{speed}$ aussortiert, die nach dem Bewegungsmodell in eine Kollision führen. Dadurch werden sie nicht für das spätere Optimierungsproblem verwendet und sind wie die Geschwindigkeiten außerhalb des Dynamic Windows irrelevant.

- soft constraints:
  Unter *soft constraints* versteht man die Parameter, die man bei der Navigation optimieren möchte. Darauf wird im weiteren Verlauf detaillierter eingegangen.

Der implementierte Algorithmus kann im Pseudocode in Alg. 4.1 zusammengefasst werden. Dabei besteht das Prinzip aus den Arbeiten von Fox et al. ([Die97]), Arras et al. ([APT+02]) und Maroti et al. ([MSK+13]).

Algorithmus 4.1: *Kollisionsvermeidungsalgorithmus basierend auf [Die97], [APT+02] und [MSK+13]*

```
1  algorithm calculate_movement_control(u_{speed,k-1}, x_r, x_goal, {p_{i,k-1}}, Δt)
2  input: control u_{speed,k-1}, robot pose x_r, goal pose x_goal, laserscan points in
       robot frame {p_{i,k-1}}, (time period Δt)
3  output: control u_{speed,k}
4
5  begin
6
7    p_coll ← add close map points and {p_{i,k-1}}
8
9    if p_coll in critical area
10     u_{speed,k} ← (0,0)^T
11     return u_{speed,k}
12   endif
13
14   generate dynamic window using accleration limits
15   update dynamic window using goal distance
16   update dynamic window using clearance
17
18   sample radial paths
19   foreach radial path
20     get max admissible speed
21   endfor
22   update dynamic window using max admissible speed from radial paths
23
24   sample admissible speeds from dynamic window
25   weight speed samples
26
27   return speed sample with largest weight
28 end
```

Die Punktwolke $\mathbf{p}_{coll}$ ist eine Zusammensetzung aus den Laserdistanzdaten und lokalen Karteninformationen[2]

In diesem Algorithmus repräsentiert die zweidimensionale Punktwolke $\mathbf{p}_{coll}$ die Umgebung und stellt die Welt in dem Verarbeitungszeitpunkt dar, d.h. alle Berechnungen wie z.B. Kollisiondistanzen auf einem Pfad oder frontale/seitliche Abstände zu Objekten werden auf Grundlage dieser Punktwolke getroffen. Diese wird im Fall vom Testaufbau aus Kap. 5.4 aus den Laserdistanzdaten und lokalen Karteninformationen[3] zusammengesetzt. Dabei wird automatisch angenommen, dass sich die dadurch dargestellten Objekte innerhalb des Zeitraums Δt sich nicht räumlich verändern, also statisch sind. Dynamische Objekte werden demnach nur betrachtet, wenn die Abtastrate $1/\Delta t$ ausreichend hoch ist, sodass die Dynamik der Objekte durch ihre stückweise stationäre Abtastung

[2] Die Zellen der metrischen Karte werden als ihre Eckpunkte mit in die Punktwolke aufgenommen. Es werden dabei nur Zellen in der näheren Umgebung des Roboters verwendet.
[3] Die Zellen der metrischen Karte werden als ihre Eckpunkte mit in die Punktwolke aufgenommen. Es werden dabei nur Zellen in der näheren Umgebung des Roboters verwendet.

erfasst werden kann.

Im nächsten Schritt wird die Punktwolke dazu verwendet kritische Hindernisse zu identifizieren. In Zeile 9 von Alg. 4.1 wird dies über „critical area“ abstrahiert. Da die vorliegenden Punktdaten im Koordinatensystem vom Roboter vorliegen und ein rechteckiger Umriss angenommen wird, ist der Vergleich wie folgt:

$$\begin{aligned} |p_{yi,coll}| &< (l_{robot} + \epsilon) \\ |p_{xi,coll}| &< (w_{robot} + \epsilon) \end{aligned} \tag{4.1}$$

$$p_{yi,coll} < (l_{robot} + \epsilon_{front}) \tag{4.2}$$

$p_{xi,coll}$ und $p_{xi,coll}$ sind die x,y-Koordinaten der Kollisionspunkte und l_{robot} bzw. w_{robot} die Länge und Breite des Roboters. ϵ und ϵ_{front} definieren entsprechende Sicherheitsabstände. Glg. 4.1 garantiert zunächst, dass keine Objekte im Umriss und ihre nähere Umgebung sind. Dieser Fall sollte im idealen System nie auftreten. In der Praxis sind Lokalisierungssprünge bzw. Fehllokalisierungen möglich, sodass dieser Fall abgedeckt werden muss. Glg. 4.2 definiert einen frontalen Mindestabstand zu Objekten. Dieser ist ein zusätzlicher Sicherheitsmechanismus, um die Dynamik in der Vorwärtsbewegung zu berücksichtigen und ist für die generelle Funktion des Algorithmus nicht zwingend notwendig. Diese Abfragen werden ebenfalls im *Security Filter Module* (siehe Abb. 4.1) durchgeführt. Der Unterschied ist, dass die Verarbeitung im Vergleich zum *Collision Avoidance Module* hochfrequentiger, bzw. mit jedem Laserscan, getriggert wird.

4.2.2 Bestimmung der zulässigen Geschwindigkeiten

Bei unkritischen Konstellationen der Punktwolke folgt der nächste Schritt: Hier wird überprüft, welche Geschwindigkeitsvektoren $\mathbf{u}_{speed}$ erlaubt sind. Der Suchraum wird hier durch folgende Vorraussetzungen eingeschränkt, die den Kern der *hard constraints* des DWAs bilden:

1. Begrenzte Fahrgeschwindigkeit
2. Begrenzte Beschleunigung
3. Kollisionsgegenstände

Der erste Punkt definiert somit ein Geschwindigkeitsgebiet $\mathcal{G}_s$, welches bei jedem Sampling gleich bleibt.

$$V_s = \{(u_v, u_\omega) | u_v \in [v_{min}, v_{max}] \wedge u_\omega \in [\omega_{min}, \omega_{max}]\} \tag{4.3}$$

Der zweite Punkt geht auf die beschränkte Beschleunigung ein. In einem festem Zeitraum Δt sind dadurch nur die Beschleunigungen möglich, die von der Anfangsgeschwindigkeit entsprechend abweichen, d.h.

$$V_d = \{(u_v, u_\omega) | u_v \in [v_0 - \dot{v}_{min} \cdot \Delta t, v_0 + \dot{v}_{max} \cdot \Delta t] \wedge u_\omega \in [\omega_0 - \dot{\omega}_{min} \cdot \Delta t, \omega_0 + \dot{\omega}_{max} \cdot \Delta t]\} \tag{4.4}$$

Der dritte Punkt beschreibt die Einschränkung durch Kollisionsobjekte. Das Ziel hierbei ist alle Geschwindigkeiten zu verbieten, die zu einer Kollision führen. Dazu gehören Bewegungen, die in diesem Zeitschritt zwar nicht mit der Punktwolke zusammenstoßen, aber im näheren Zeitraum mit der maximalen Bremsbeschleunigung nicht mehr davor abgebremst werden können.

$$V_a = \{(u_v, u_\omega) | u_v \leq \sqrt{2 \cdot dist(u_v, u_\omega) \cdot \dot{v}_{min}} \wedge u_\omega \leq \sqrt{2 \cdot dist(u_v, u_\omega) \cdot \dot{\omega}_{min}}\} \tag{4.5}$$

Man beachte, dass v_{min}, v_{max}, $\dot{v}_{min}$, $\dot{v}_{max}$, ω_{min}, ω_{max}, $\dot{\omega}_{min}$ und $\dot{\omega}_{max}$ feste konfigurierbare Parameter sind, um die Dynamik des Systems zu beschreiben. Der Durchschnitt dieser drei Gebiete bildet den Optimierungsraum für den nächsten Schritt, also

$$V_r = V_s \cap V_d \cap V_a. \tag{4.6}$$

Abb. 4.3 zeigt eine beispielhafte Visualisierung der Unterräume des Geschwindigkeitsraums. Man erkennt hier die Objekte indirekt in V_a. Es ist anzumerken, dass die Visualisierung nur die ideale Menge V_a zeigt und diese in der Praxis nicht in ihrer Gesamtheit ausgerechnet werden. Im Gegensatz zu V_s und V_d ist diese nicht immer eindeutig parametrisierbar und von den Werten $\mathbf{p}_{coll}$ abhängig, die eine Diskretisierung der kontinuierlichen Umgebung sind.

Um nun die Menge V_a effizient anzunähern wurde die Methode des *radial sampling*s aus der Arbeit von Maroti et al. ([MSK+13]) angewendet. Dieser nutzt die Eigenschaft des Bewegungsmodells aus Kap. 2.1 aus, dass die Trajektorie des Roboters ausschließlich aus Kreissegmenten besteht, die mit einer bestimmten Geschwindigkeit abgefahren werden können. Wenn auf einem Radius die maximale Fahrdistanz bestimmt werden kann, dann gilt diese für alle Fahrgeschwindigkeiten auf diesem Radius. D.h. es werden zunächst eine feste Anzahl Radien für den Unterraum $V_s \cap V_d$ gesampelt, für die die Kollisionsdistanz $dist(u_v, u_\omega)$ berechnet wird. Je mehr Radien verwendet werden, desto vielfältiger sind die Navigationsmöglichkeiten. Dadurch sind präzisere Fahrmanöver möglich, aber dies erhöht zugleich auch den Rechenaufwand.

In Abb. 4.4 ist ein Muster der gesampelten Geschwindigkeitsvektoren abgebildet. Man erkennt, dass sich die abgetasteten Werte eines Radius auf einer Geraden zum Ursprung

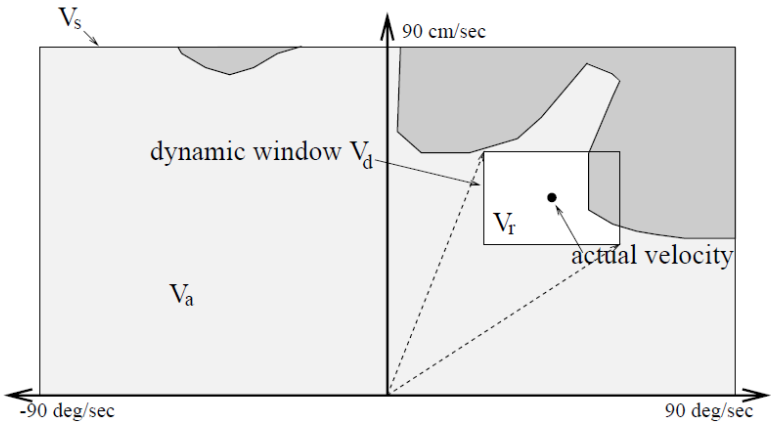


Abbildung 4.3: *Diagramm des Dynamic Windows im Geschwindigkeitsraum [Die97]. Auf der x-Achse ist die Rotationsgeschwindigkeit u_ω und auf der y-Achse ist die Translationsgeschwindigkeit u_v abbgebildet.*

befinden. Auch in einem Diagramm mit u_ω auf der x-Achse und u_v auf der y-Achse ist dies der Fall. Der Grund dafür ist die lineare Abhängigkeit von u_ω und u_v für konstante Radien (siehe Glg. 2.3).

4.2.3 Berechnung der Kollisionsdistanzen

Die Berechnung der Kollisionsdistanz $dist(u_v, u_\omega)$ reduziert sich durch die Methode von Maroti et al. auf $dist(R = \frac{u_v}{-u_\omega})$, d.h. es werden nur Distanzen für bestimmte Radien berechnet.

Da die Umgebungsinformation in der Punktwolke $\mathbf{p}_{coll}$ enthalten ist, werden die einzelnen Distanzen der Punkte zum Umriss des Roboters berechnet und ihr Minimum für die maximale Fahrdistanz verwendet.

$$dist(R = \frac{u_v}{-u_\omega}) = \min_i arc(R, \mathbf{p}_{i,coll}) \tag{4.7}$$

Dabei bezeichnet $arc(R, \mathbf{p}_{i,coll})$ aus Glg. 4.7 den Kreisbogen, den der Roboter zurücklegen darf bis der Punkt $\mathbf{p}_{i,coll}$ mit dem Umriss des Roboters in Kontakt kommt mit $(R, 0)^T$ als Rotationspunkt.

Bei $R = \frac{u_v}{-u_\omega}$ rotiert nicht nur die Roboterpose um den Rotationspunkt, sondern auch die Punkte, die mit dem Umriss kollidieren können. Abb. 4.5 zeigt die Gebiete im Roboterkoordinatensystem, die auf mögliche Kollisionen überprüft werden müssen. Dazu werden iterativ die Punkte in $\mathbf{p}_{coll}$ darauf überprüft, ob sie im kritischem Gebiet sind. Der ana-

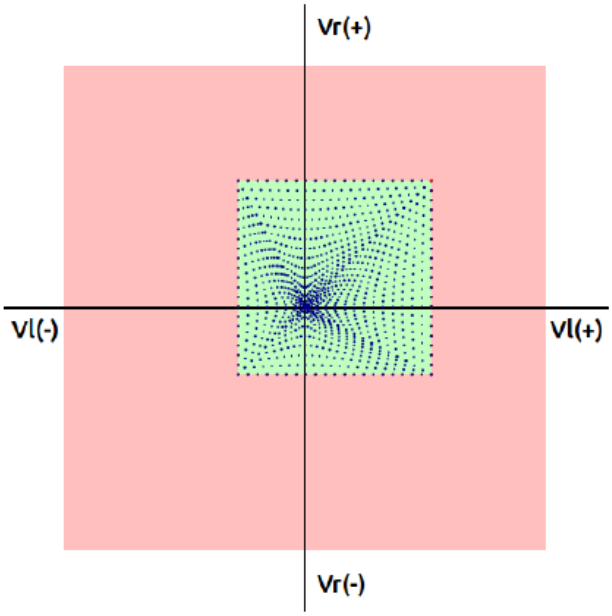


Abbildung 4.4: *Muster des radial samplings [MSK+13] eines Systems mit Differentialantrieb. vl und vr bezeichnen entsprechend Geschwindigkeit des linken und rechten Antriebs.*

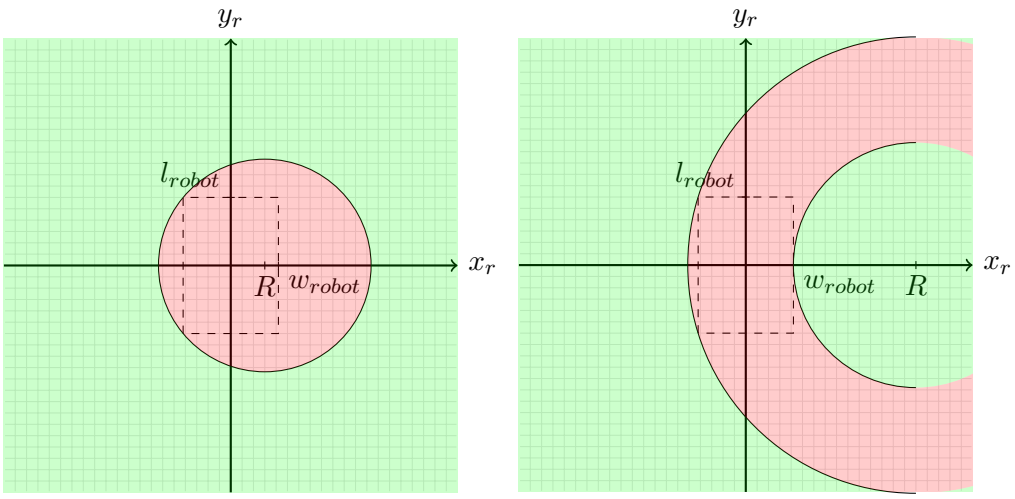
lytische Weg besteht darin den Abstand vom euklidischen Abstand vom Kollisionspunkt zum Rotationspunkt $(R, 0)^T$ zu berechnen und mit den kritischen Abständen des rechteckigen Umrisses zu vergleichen. Dabei variieren die kritischen Abstände je nachdem wo sich der Rotationspunkt befindet. In Abb. 4.5a ist der Rotationspunkt innerhalb des Umrisses, also $|R| \leq w_{robot}$. Es ergeben sich nur zwei Gebiete, zwischen denen unterschieden werden muss. Für eine Kollsion ist die folgende Ungleichung wahr:

$$\left\| \mathbf{p}_{i,coll} - (R, 0)^T \right\| \leq \sqrt{(|R| + w_{robot})^2 + l_{robot}^2} \tag{4.8}$$

Bei Abb. 4.5b sind es hingegen drei Gebiete. Dabei ist die charakteristische Ungleichung für Kollisionen:

$$||R| - w_{robot}| \leq \left\| \mathbf{p}_{i,coll} - (R, 0)^T \right\| \leq \sqrt{(|R| + w_{robot})^2 + l_{robot}^2} \tag{4.9}$$

Sind die Punkte, die kollidieren können, erkannt, können die Distanzberechnungen durchgeführt werden. Zunächst werden die Überschneidungspunkte berechnet vom Rechteck des Roboterumrisses und dem Kreis, welcher aus $\mathbf{p}_{i,coll}$ und den Rotationspunkt als Zentrum resultiert. Dies kann über die Gleichungen aus der Arbeit von Arras et al. ([APT+02]) gemacht werden. Danach wird der in Rotationsrichtung nächste Überschnei-



(a) $|R| > w_{robot}$ **(b)** $|R| \leq w_{robot}$

Abbildung 4.5: *Kritische Gebiete mit Kollisionspotential für einen radialen Pfad.*

dungspunkt gewählt, um den Kreisabschnitt zum Umriss zu berechnen. Über den Dreisatz wird schließlich die Pfadlänge des Roboters bestimmt.

$$arc(R, \mathbf{p}_{i,coll}) = \frac{R}{\|\mathbf{p}_{i,coll} - (R,0)^T\|} \cdot \widehat{AB} \tag{4.10}$$

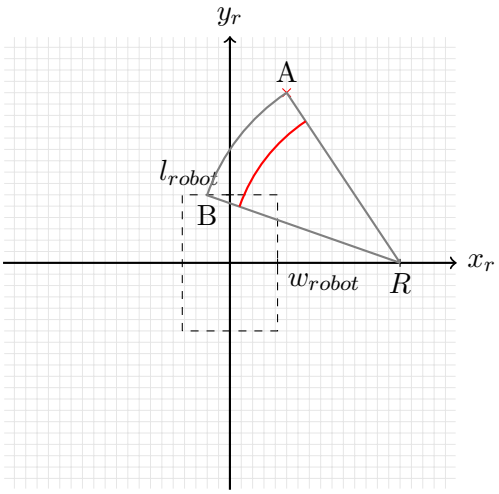


Abbildung 4.6: *Visualisierung des Kollisionspfads in Bezug auf den Kollisionspunkt und den Roboterpfad.*

Die Bestimmung der Kollisionsdistanz wird in Abb. 4.6 dargestellt. Der rote Kreisbogen stellt $arc(R, \mathbf{p}_{i,coll})$ dar. Die Bestimmung des nächsten Punktes B aus allen möglichen

Überschneidungspunkten kann entweder über eine Heuristik (z.B. euklidische Distanz $\overline{AB}$) bestimmt werden oder über die Längenbestimmung aller Kreisbögen. In dieser Arbeit wurde dies über Letzteres realisiert.

Es ist zu erkennen, dass die Berechnung der Kollisionsdistanz deutlich vom Umriss des Roboters abhängt. Bei ihrer manuellen Berechnung sind dabei viele Fallunterscheidungen notwendig, auch bei einfachen Formen wie dem Rechteck. Da man dieses Problem zu einem geometrischen Problem reduzieren kann, d.h. Überschneidungspunkte von Kreisen und dem Umriss, können vorhandene Subroutinen aus vorhandenen Bibliotheken verwendet werden (z.B. CGAL), um diesen Schritt vereinfacht und möglicherweise optimierter durchzuführen.

4.2.4 Auswahl der optimalen zulässigen Geschwindigkeit

In Kap. 4.2.2 wurden die zulässigen Geschwindigkeiten auf Basis von Geschwindigkeits- und Beschleunigungsgrenzen unter Berücksichtigung der Kollisionsgegenstände und ihr maximaler Pfad bis zur Kollision ausgewählt. Das Resultat ist die Menge V_r, aus der die „optimale“ Geschwindigkeit bestimmt werden muss, die letztendlich die Bewegung im nächsten Zeitschritt bestimmt. Im ursprünglichen DWA ([Die97]) und den darauf aufbauenden Algorithmen ([BK99], [APT+02], [MSK+13] und [PS03]) bildet die sogenannte *objective function* die Metrik für die Qualität der Geschwindigkeiten. Obwohl die tatsächliche Realisierung dieser Metrik sich über die Entwicklung des Algorithmus und den vorhandenen Umgebungsinformationen unterscheidet, kann man ihre Zusammensetzung in folgende Anteile (siehe auch Abb. 4.7) zusammenfassen:

- **Ausrichtung** zum zielführenden Pfad:
  In der Arbeit von Fox et al. ([Die97]) wurde dieser Anteil als einfacher Winkel zwischen Zielposition und momentaner Ausrichtung definiert. Das Prinzip des line of sight (LOS) ist in der Navigation von Raketen und Flugzeugen ein fundamentaler Ansatz ([KG09]). Wenn man die Navigation als Regelungsproblem mit einem Sollpfad betrachtet, hat dieses Prinzip seine Nachteile[4]. Dennoch ist bei geeigneter Parameterwahl und ausreichend hoher Aktualisierungsrate das Regelverhalten stabil. Bei Integration globaler Informationen wird statt der Zielposition, der zielführende Pfad verwendet. Diese wird in der Literatur als *navigation function* (Navigationsfunktion) bezeichnet ([BK99]).

  Dieser Anteil ist essentiell für die Initiierung und Garantierung einer zielführende Bewegung. Bei der lokalen Planung ist das nicht immer gewährleistet (lokale Minima).

[4] z.B. Überschwingverhalten. Das Regelungsprinzip ist vergleichbar mit einem P-Regler.

- **Fortschritt** auf dem zielführenden Pfad:
  Bei Fox et al. ([Die97]) wird ausschließlich die Geschwindigkeit als Maß verwendet, um den Fortschritt in Richtung Ziel zu bewerten. Brock et al. ([BK99]) integriert hier zusätzlich Karteninformationen, um den Fortschritt genauer bewerten zu können.

  Diese Gewichtung soll schnellere Bewegungen favorisieren. Obwohl man sich zum Ziel optimal ausrichten kann, ist es ebenfalls notwendig sich zum Ziel hinzubewegen.

- **Vermeidung** von Objekten:
  Die ursprüngliche Version von Fox bezeichnet diesen Teil der Funktion als *clearance*, welcher die Distanz zu naheliegenden Objekten bestimmt. D.h. je weiter entfernt man zum Objekt ist, desto besser fällt das Gewicht der Funktion aus. Bei den hybriden Varianten ([BK99] und [MSK+13]) wurde das indirekt in der Navigationsfunktion berücksichtigt. Dies bedeutet aber auch, dass alle Kollisionsinformationen in der Karte enthalten sind. Abhängig von Echtzeitanforderungen, Kartenaktualisierungsrate und dynamischen Objekten ist dies jedoch nicht immer garantiert.

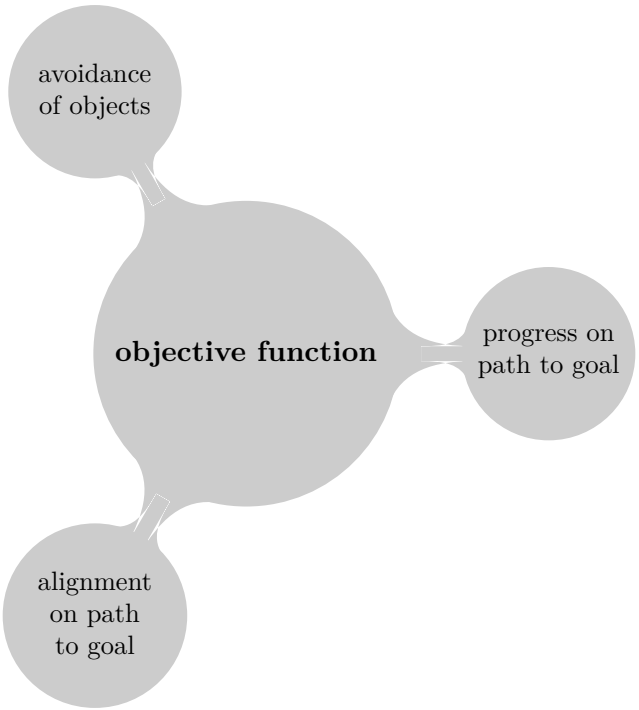


Abbildung 4.7: *Zusammensetzung der objective function aus seinen essentiellen abstrahierten Anteilen.*

Die Metrik, die in dieser Arbeit für das Optimierungsproblem verwendet wurde, ist aus

der Arbeit von Fox et al. ([Die97]):

$$G(u_v, u_\omega) = \sigma(\alpha \cdot head(u_v, u_\omega) + \beta \cdot clear(u_v, u_\omega) + \gamma \cdot vel(u_v, u_\omega)) \tag{4.11}$$

und berechnet für alle validen Geschwindigkeitstupel einen Gewichtswert. Das Tupel mit dem höchsten (bzw. niedrigsten) Gewicht[5] wird schließlich als Ausgang verwendet. Genauere Angaben zu den Gewichtsfunktionen finden sich in Kap. A.2.

Der Grund dafür, dass die Gewichtung des ursprünglichen Algorithmus verwendet wurde und nicht die Gewichtung über die Navigationsfunktion ([BK99] oder [MSK+13]), obwohl eine Rasterkarte der Umgebung vorhanden ist (siehe Kap. 3.3), liegt in der Kartenauflösung. Da die Implementierung der Karte in dieser Arbeit einfach gehalten wurde, ist die Rechenkomplexität der Kartenaktualisierung nicht optimiert und es wurde sich auf eine niedrige Auflösung ($0.1m$ Zellbreite) geeinigt. Die verschiedenen Geschwindigkeitspaare sind im Allgemeinen nicht unterscheidbar, da diese sich innerhalb einer Zelle befinden. Daher wurde stattdessen der alternative Ansatz ohne Rasterkarte verwendet.

4.2.5 Zusätzliche Modifikationen am Algorithmus

Obwohl die in Kap. 4.2.2 und Kap. 4.2.4 behandelten Bedingungen für die Implementierung der Kollisionsvermeidung ausreichend sind, werden zusätzliche Eigenschaften in Algorithmus 4.1 berücksichtigt, auf die im Folgenden weiter eingegangen werden.

- Geschwindigkeitsbegrenzung bei nahem Ziel:
  Um eine präzisere Ansteuerung der Zielkoordinaten zu gewährleisten, wird die Geschwindigkeit in der Nähe des Ziels gedrosselt. Dabei dient die euklidische Distanz der Roboterposition zur Zielposition als Maß.

- Geschwindigkeitsbegrenzung bei nahen Objekten:
  Bei nahen Objekten wird die Geschwindigkeit gedrosselt, um zum einen das Risiko für eine Kollision niedrig zu halten und zum anderen das Umfahrmanöver flexibel zu gestalten.

- Rotation am Ziel:
  Da die Metrik in Glg. 4.11 nur die Position des Ziels berücksichtigt, muss für eine Bewegung von einer Pose zu einer anderen Pose die Zielorientierung zusätzlich angefahren werden. Dies wurde dadurch realisiert, dass der Roboter beim Erreichen der Zielposition mit einer Toleranz solange rotiert bis die Zielorientierung erreicht wurde. Dies geschieht unter der Annahme, dass die *hard constraints* erfüllt sind.

[5] je nach Implementierung

Die Begrenzungen sind als Modifikation des Dynamic Windows (*hard constraints*) implementiert.

4.3 Planung auf Basis der Rasterkarte

Die Rasterkarte wird mit Sensordaten aktualisiert und speichert globale Umgebungsinformationen, daher ist diese auch für eine globale Pfadplanung geeignet. Obwohl ihr Einsatz aus Laufzeitgründen in dieser Arbeit nicht aktiv eingesetzt werden kann, bildet die Implementierung der Planungsmethoden hier die Grundlage für spätere Integrationen in den Planungsalgorithmus. Z.B. kann nach Optimierung der Kartenaktualisierung eine höhere Auflösung bei gleichem Rechenaufwand betrieben werden. Aus der Karte kann die *navigation function* berechnet werden, die in der Kollisionsvermeidung angewendet werden kann (siehe [BK99] und [MSK+13]).

Die Planung erfolgt auf Grundlage der Rasterkarte, auf der graphen-basierte Algorithmen angewendet werden, um Kostenfunktionen über den Navigationsraum zu erstellen (u.a. die Navigationsfunktion). Dadurch unterliegt die Kostenfunktion Diskretisierungsartefakten, die anwendungsbezogene Nachteile mit sich bringen (siehe Kap. 4.2.4). Dennoch können diese Artefakte bei genügend hoher Auflösung vernachlässigt werden. Ansonsten wäre eine Interpolation der diskreten Kostenfunktion eine weitere mögliche Lösung.

Die Umsetzung der Navigationsfunktion wurde in dieser Arbeit dreiteilig nach dem Vorbild von Lu aufgebaut ([Lu14, App. A2.1]). Jeder der folgenden Unterpunkte erzeugt zunächst eine Karte mit Gewichten in ihrer Zelle (*costmap*), die zusammen aufsummiert die zellabhängige Navigationsfunktion bilden.

1. Distanz zum Ziel
2. Distanz zu Objekten
3. Distanz zum Pfad

Als Ansatz wurde eine Breitensuche angewendet, um einen zielführenden Pfad zu generieren. Zunächst wurde die binäre Rasterkarte mit der Breite des Roboters dilatiert (*inflated occupancy gridmap*). Die verbliebenen freien Zellen bilden die Knoten in dem Suchgraphen, auf dem der Algorithmus angewendet wird. Dabei sind die Zellen in dem Graphen miteinander verbunden. Diese befinden sich in einer Achter-Nachbarschaft (*moore neighbourhood*) zueinander. Für die Kantengewichtung wurde die Distanz der Zellzentren verwendet. Jeder freien Zelle wird ausgehend vom Ziel ein Wert zugeordnet, der die niedrigste Summe der Kantenlängen entspricht.

Der Unterraum, der durch die Dilatation entsteht, ist eine Untermenge des Navigationsraums, da diese die Orientierung des Roboters nicht berücksichtigt. Um den genauen Navigationsraum abbilden zu können, müssen orientierungsabhängig dilatierte Rasterkarten erstellt werden, die den Suchraum um eine Dimension erweitern und den Aufwand erheblich vergrößern. Daher wurde das vereinfachte Problem verwendet. Der offensichtliche Nachteil dieser Vereinfachung ist, dass Pfade nicht gefunden werden, die vom Roboter abgefahren werden können wie z.B. enge Türen etc. .

Die *inflated occupancy gridmap* wird ebenfalls als Ausgangspunkt für die nächste *costmap* verwendet. Diese berechnet die Distanz zu naheliegenden Objekten, d.h. freie Zellen, die unterhalb einer Mindestdistanz zu einer belegten Zelle sind, bekommen eine Gewichtung. Nach der Distanzkonvention der ersten *costmap* werden nahe Zellen stärker gewichtet als ferne Zellen.

Ausgehend von der Startzelle kann mit der ersten Kostenkarte über einen Gradientenabstieg der Pfad zur Zielzelle ermittelt werden. Diese Abfolge der Zellen ist nun der Navigationspfad. Für die letzte *costmap* werden die Zellen dieses Pfades mit dem Gewicht Null belegt und mit einem *wavefront*-Algorithmus ähnlich zur ersten *costmap* werden die Gewichte berechnet.

KAPITEL 5

Implementierung

In diesem Kapitel wird die Umsetzung der in Kap. 3 und 4 beschriebenen Algorithmen näher erläutert. Ein wesentlicher Schwerpunkt dabei ist die Modularität der Funktionalitäten, d.h. die Gesamtfunktion des Systems wird in trennbare Unterfunktionen aufgeteilt, die unabhängig voneinander verwendet werden können. Wie in Kap. 1.2 angesprochen, werden diese Module als Bibliotheken in der Softwareumgebung *VEROSIM* implementiert.

Anschließend wird auf das Setup eingegangen, mit der die Funktionalität validiert und ausgewertet wurde. Zum einen wurde die Simulationsumgebung von *VEROSIM* und zum anderen wurde der reale Prototyp verwendet.

5.1 VEROSIM

Das hier verwendete Softwareframework VEROSIM wurde mit der Motivation entwickelt Daten zu akquirieren, prozessieren, visualisieren, simulieren und zu regeln ([VER18]). Sie bildet die Basis für die hier implementierten Algorithmen. Neben ihrer Mikrokernelarchitektur und weiteren Features[1] eignet sich VEROSIM besonders für die Entwicklung, dadurch dass das simulation-based prototyping möglich ist. So können in der Simulation gewisse Teilprozesse verifiziert und analysiert werden, wobei die Simulation mit realitätsnahen Rahmenbedingungen konfigurierbar ist. Das Debuggen kann durch die Visualisierungsmöglichkeiten erheblich erleichtert werden. Hinzu kommt die modulare Struktur: Dadurch sind Module aus anderen Plugins in Kombination miteinander anwendbar, sodass Redundanzen in den Implementierungen gering gehalten werden können. Konkrete Beispiele in dieser Arbeit sind z.B. die Verwendung der Visualisierungstools, die Dynamiksimulation, Gamepadcontroller etc. . Bei einer erfolgreichen Simulation kann die

[1] siehe [VER18] für weitere Details

Datenprozessierung der Simulation auf die reale Hardware portiert werden, indem die entsprechenden Module für Dateneingänge bzw. Datenausgang ausgetauscht werden. Somit ist die Algorithmenentwicklung unabhängig vom unterliegenden System.

Abb. 5.1 zeigt ein Beispiel von der Modellierungsoberfäche „IO-Board“ der VEROSIM-Umgebung. Die Module sind als Blackbox mit I/O-Ports dargestellt, die untereinander verbunden werden können. Dabei wird die Prozessierung der Daten nach dem Signal-Slot-Konzept, also ergebnisorientiert, getriggert. Diese Datenverarbeitungsebene ist auch an der graphischen Simulation angebunden, sodass Prozessschritte in einer 3D Umgebung visualisiert werden können.

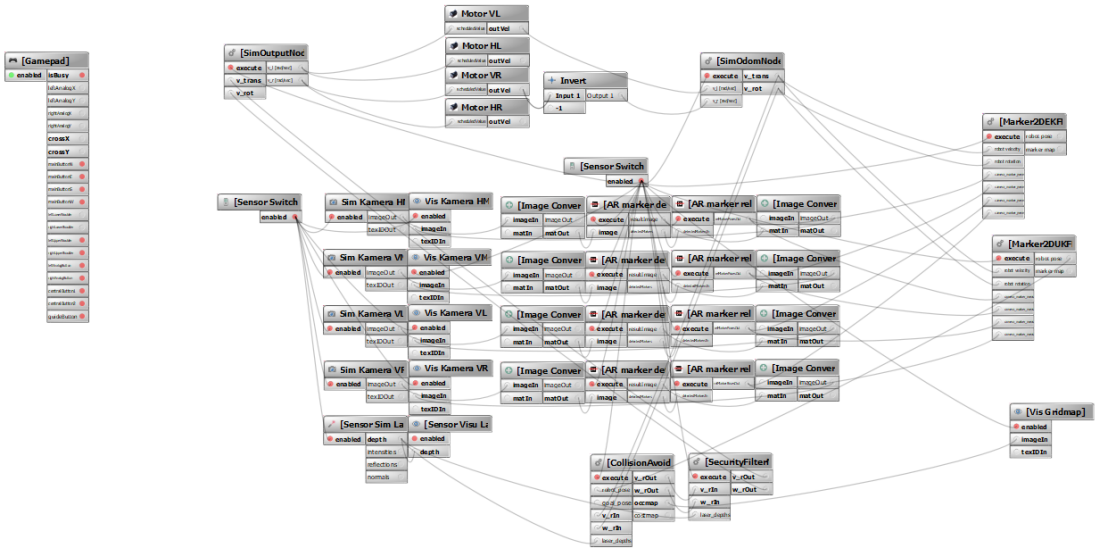


Abbildung 5.1: *IO-Board in VEROSIM. Die Struktur zeigt den Aufbau des simulierten teilautonomen System von der direkten Verarbeitung der Sensordaten bis hin zu den Steuerungsmodulen.*

5.2 Implementierte Plugins

Für die Arbeit wurde ein existierendes Plugin erweitert und vier weitere Plugins erstellt. Dabei wurde die institutsseitige Namenskonvention eingehalten, die VEROSIM-spezifische bzw. -unspezifische Funktionen aufteilen. Die Plugins mit dem Präfix „VSLib“ implementieren Kernfunktionen der Algorithmen und enthalten entsprechende Klassenstrukturen, die extern verwendet werden. Die Plugins mit dem Präfix „VSPlugin“ hingegen stellen die Schnittstelle zum VEROSIM-Interface bereit, d.h. Userinteraktionen und Visualisierung etc. sind in dieser Bibliothek integriert.

In Tab. 5.1 ist eine Übersicht der Plugins mit ihren implementierten Funktionalitäten in den einzelnen Klassenstrukturen dargestellt. Dabei werden die Funktionalitäten in Stichpunkten generalisiert und evt. Helperfunktionen nicht seperat erwähnt. Außerdem wird die C++-Standardbibliotheken bis zu C++11 als Implementierungsstandard angenommen. Man erkennt, dass bei einer Ausweitung der Funktionen der Plugin VSLibMarkerLocalization, welcher hier als Gruppierung der Lokalisierung und Kartierung dient, noch einmal unterteilt werden kann. Natürlich sind die Plugins auch von anderen VEROSIM Plugins abhängig wie z.B. VSS, VSDIO, VSD, VSLibRenderGL etc. , darauf wird aber hier nicht näher eingegangen.

Pluginname	**Entity / Klasse**	**Externe Bib**	**Funktionalität**
VSLibARMarker (modifiziert)	RelativePoseNode	– OpenCV (ArUco Library)	– Kamerakalibrierung – Posenprojektion und Transformation ins 2D-Roboterkoordinatensystem
VSLib-MarkerLocalization	EKF2DInstance	– Eigen Libary	– Prädiktionsschritt – Korrekturschritt – Parsen der csv-Datei für die statische Karte
	UKF2DInstance	– Eigen Libary	– Prädiktionsschritt – Korrekturschritt – Parsen der csv-Datei für die statische Karte
	Gridmap	-	– Occmap Generation und Update – Costmap Generation und Update – (Hashfunktion für std::pair)

VSLib-MobileRobot-Navigation	CollavDWA	-	– Berechnung der optimalen Geschwindigkeitsregelgröße basierend auf der DWA Metrik – Posenprädiktion für Geschwindigkeitspaare – relative Punktwolkenprädiktion für Geschwindigkeitspaare
VSPlugin-MarkerLocalization	2DEKFNode	-	– Interface zu EKF2DInstance – Dynamisches I/O-Board
	2DUKFNode	-	– Interface zu UKF2DInstance – Dynamisches I/O-Board
	MappingNode	-	– Interface zu Gridmap
	NodeVisu	-	– Visualisierung von EKF2DInstance – Visualisierung von UKF2DInstance
	SensorMergeNode	-	– Fusion mehrerer Kamerainputs zu einem gemeinsamen Output – Dynamisches I/O-Board

	SimOdomNode	-	– Konvertierung von rechter/linker Radgeschwindigkeit zu Translation- und Rotationsgeschwindigkeit
VSPlugin-MobileRobot-Navigation	CollisionAvoidance Node	-	– Interface zu CollavDWA – Integration lokaler Gridmappunkte in die Laserscanpunktwolke
	NodeVisu	-	– Visualisierung von CollavDWA
	SecurityFilter Node	-	– Überprüfung kritischer Messungen und bei Bedarf Notbremsung
	SimOutputNode	-	– Konvertierung von Translation- und Rotationsgeschwindigkeit zu rechter/linker Radgeschwindigkeit

Tabelle 5.1: *Übersicht der in der Arbeit neu implementierten Funktionalitäten.*

In Abb. 5.2 ist die Kommunikationsstruktur der Plugins untereinander dargestellt. Dort wird ebenfalls veranschaulicht, welche Komponenten bei der halb-autonomen Navigation wiederverwendet und welche speziell angefertigt wurden.

5.3 Simulation des Roboters in VEROSIM

Zunächst wurden die Plugins und ihre Unterfunktionen schrittweise in einer Dynamiksimulation validiert und optimiert. Dazu existiert ein digitaler Zwilling in der Umgebung, das die gleichen geometrischen Eigenschaften besitzt wie das reale Modell. Die Dynamik des Systems wurde vereinfacht dargestellt: Der Körper des Roboters wird als Quader angenähert und die Räder werden als Ellisoiden modelliert. Die Massenträgheit des Ro-

boters befindet sich hier im Quadermodell. Kamera- und Lasersensoren wurden von dem Plugin „VSLibSensor“ modelliert und deren Parameter entsprechend an unsere Situation angepasst.

Eine wichtige Anmerkung ist, dass der Scheduler die simulierte Zeit so anpasst, dass alle Module die benötigte Zeit bekommen, um ihre Berechnungen durchzuführen. Demnach ist die Echtzeitfähigkeit des Systems in der Simulation garantiert. Wenn die Rechenzeit

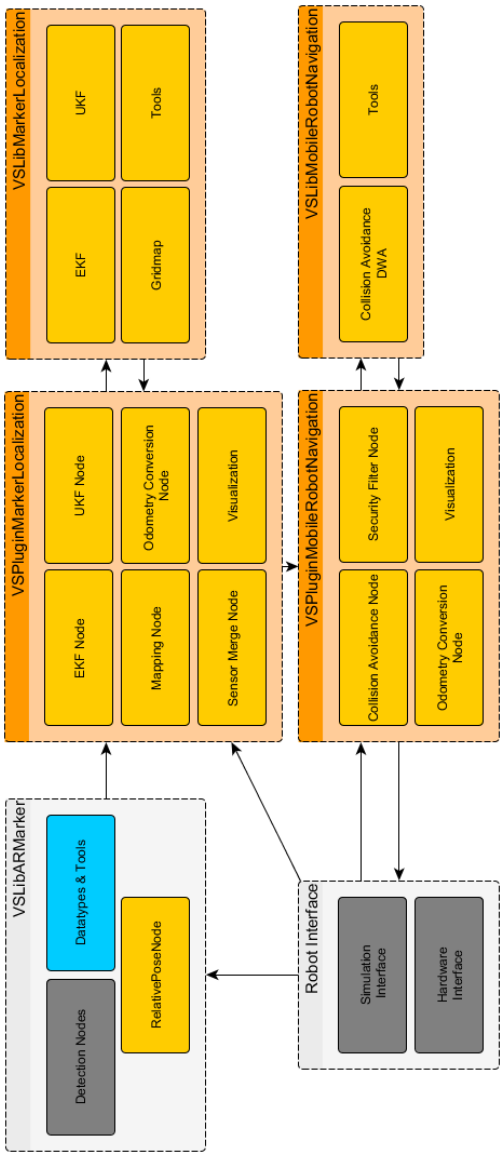


Abbildung 5.2: *Abhängigkeitsstruktur der VEROSIM Plugins untereinander. Orange markierte Komponenten wurden in dieser Arbeit neu angefertigt, blaue Komponenten wurden modifiziert und graue Komponenten sind bereits vorhanden*

länger ist als ein Simulationszeitschritt, wird dieser solange angehalten bis die Ausführung beendet ist. Der durchschnittliche Echtzeitfaktor einer Simulation beträgt bei den Testszenarios aus Kap. 6.1 - Kap. 6.3 ca. 0.3 - 0.5 im Release-Build. Es wird also erwartet, dass das System nicht echtzeitfähig ist. Einige Parameter des Schedulers sowie sonstige Einstellungen der Simulation werden in Kap. B.1 aufgelistet.

5.4 Seekur Jr Prototyp Setup

Nach der Validierung in der Simulation wurden die Plugin auf der Hardware des Seekur Jrs getestet. Der Seekur Jr besitzt einen internen Rechner, an dem die Systemkomponenten verbunden sind. In diesem Roboter wurde ein 2D-Lidarsensor von SICK mit der Bezeichnung LMS151 verwendet ([AG17]). Es wurden vier Webcams installiert, wobei drei auf der Vorderseite angebracht sind und eins an der Rückseite (Logitech HD Webcam C510 [Log10]). Eine Kennzeichnung der Sensoren am Prototypen ist in Abb. 5.3 zu erkennen. Die Webcams liefern Bilder in einer Frequenz von 30Hz mit der Auflösung 960x480. Ihre Kalibrierung erfolgt vorab mit der OpenCV Bibliothek. Die genauen Parameter können in Kap. B.4 eingesehen werden.

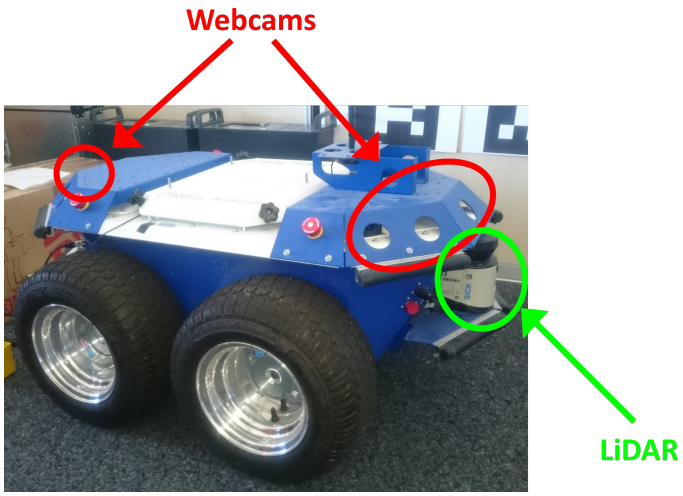


Abbildung 5.3: *Realer Prototyp des Seekur Jrs mit gekennzeichneter Sensorik.*

Die Anbindung an das Fahrwerk wird über die Software Development Toolkit (SDK) namens ARIA (Advanced Robot Interface for Applications) von MobileRobots geregelt [LLC18]. Dazu wird eine Serverapplikation auf dem internen Rechner gestartet, auf die ein Plugin in VEROSIM zugreifen kann. Somit ist ein Subset der Gesamtfunktionalität von ARIA für die Steuerung des Roboters verwendbar. Dadurch kann die Datenverarbeitung des realen Roboters ebenfalls in der VEROSIM Umgebung modelliert werden.

Obwohl das VEROSIM Modell ein verteiltes System zulässtc (die Verarbeitung von Teilmodellen auf verschiedenen Rechnern) wird hier nur der interne Rechner für den gesamten Prozess verwendet, um Effekte von Kommunikationslatenzen auszuschließen. Die Überwachung des Systems erfolgt über eine Remote-Desktop-Sitzung.

KAPITEL 6

Test Szenarios und Ergebnisse

In diesem Kapitel werden die Funktionalitäten näher untersucht. Hier dient die Simulationsoberfläche von VEROSIM hauptsächlich zur Evaluation der Lokalisierungsgenauigkeit, Kartierungsgenauigkeit und Navigationsverhalten. Es können in der Simulation Modelle erstellt werden, die eine Abstraktion von realitätsnahen Szenarien darstellen. Die gesamte Navigation wird zunächst auf einzelne Komponenten untersucht. Die entsprechenden Szenarien sind auf die Funktionsweise der Komponenten angepasst.

Generell ist die Umgebung in der Simulation idealisiert dargestellt und in den ersten drei Unterkapiteln als eine vereinfachte Büroumgebung modelliert. Die Navigationsebene wird in dieser Arbeit als zweidimensionale Ebene angenommen. Daher werden die Testumgebungen mit Rücksicht auf die Modellierungsannahmen ausgewählt. Die in dieser Arbeit verwendeten Umgebungsparameter sind in Kap. B zusammengefasst.

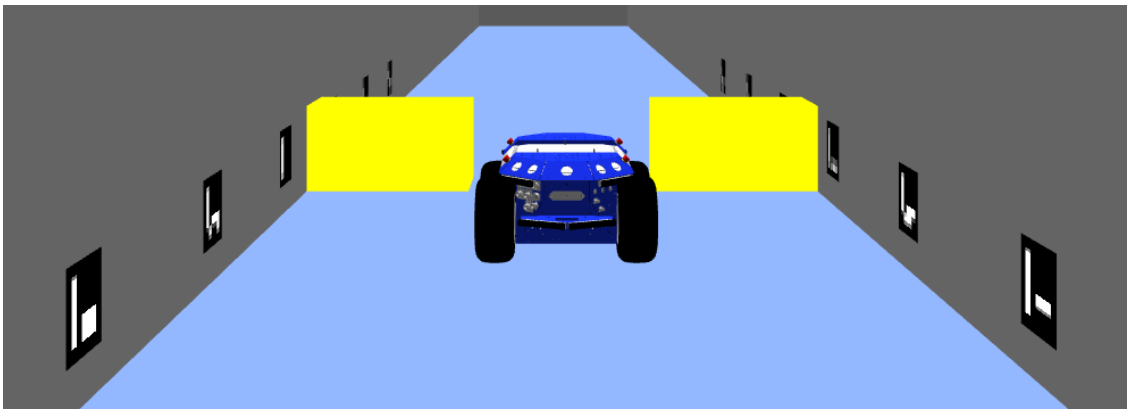


Abbildung 6.1: *Exemplarischer Fluraufbau in der Simulationsumgebung VEROSIM mit AR-Marker an den Flurwänden und gelben Boxen als Kollisionsobjekte.*

6.1 Qualität des Marker-basierten SLAMs

6.1.1 Markerlokalisierung auf der dynamischen Karte

Im ersten Schritt wurde die Lokalisierungsgenauigkeit untersucht. Dabei wurde eine geradlinige Bewegung im Flur ohne zusätzliche Objekte veranlasst. Das Bewegungssignal wurde hierbei konstant zu $\mathbf{u}_{speed} = (0.5, 0.0)^T \frac{m}{sec}$ gesetzt und für 7 sec simuliert. Der Schwerpunkt lag hierbei auf der Genauigkeit der Lokalisierung mit statisch vorgegebenen und dynamisch angelegten Markern. Die Markerkantenlänge beträgt 30 cm und die Marker sind an den Wänden mit einem Abstand von 1.5 m zueinander angebracht. Die Flurbreite beträgt 3 m. Die Kovarianzmatrix des Systemrauschen wurde zu $\mathbf{Q} = diag(0.001, 0.001, 0.004)$ gewählt und für das Messrauschen wurde $\mathbf{R} = diag(0.1, 0.1, 0.4)$ verwendet. Die initiale Roboterkovarianz wurde mit $\mathbf{P}_0 = diag(0.25, 0.25, 0.25)$ initialisiert. Die Posenfilterung der Markerkennung erfolgte mit den Parametern $\epsilon_{kante} = 30px$ und $\epsilon_{verzerrung} = 10px$.

Abb. 6.2 zeigt den traversierten Pfad von dem Roboter im ersten Versuch mit dynamisch angelegten Markern. Da ein konstanter Eingang nur mit einer Translationskomponente angelegt wurde, ist die tatsächliche Trajektorie des Roboters nahezu geradlinig[1]. Im Diagramm ist dieser Pfad als gelbe Linie dargestellt. Nimmt man diese Trajektorie als Referenz, dann erkennt man, dass die Schätzung des EKFs eine konstante Abweichung in die negative x-Komponente aufweist. Auch die UKF-Schätzung weist eine Tendenz in die gleiche Richtung auf.

Eine nähere Analyse der Schätzungen wird in den Diagrammen in Abb. 6.3 gezeigt. Da sowohl die Marker und die Roboterpose fehlerbehaftet sind und keine Informationen über exakte Positionen vorhanden sind, wirken sich die Kartierungs- und Lokalisierungsfehler additiv aufeinander aus. Insbesondere sieht man diesen Effekt in Abb. 6.3a. Dennoch befindet sich der Schätzfehler der Position bei beiden Verfahren im Zentimeterbereich, welches in diesem Versuch ca. 1 % der gefahrenen Distanz entspricht. Für die Abweichung der 2D-Position wurde der euklidische Abstand verwendet. Bei der Betrachtung der betragsmäßige Abweichung des Orientierungswinkels, lässt sich für die Schätzung des EKFs eine obere Schranke von ca. 0.13 deg und beim UKF ca. 0.45 deg definieren. In diesem Fall sind diese geringen Abweichungen nur möglich, wenn die Bewegung keine Rotationskomponente besitzt.

In diesem Versuch wurden in der Simulation 12 ArUco-Marker an den Flurwänden angebracht, dessen Pose dem Roboter vorher nicht bekannt waren. Bei der Initialisierung des Roboters befanden sich bereits Marker in seinem Sichtbereich, sodass diese im Verlauf des Versuchs in den Zustand mit aufgenommen und getrackt wurden. Die Bereiche der

[1] bei idealisierte Bedingungen

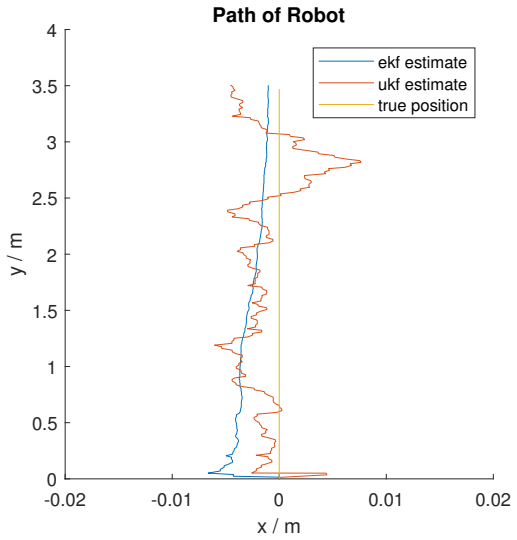
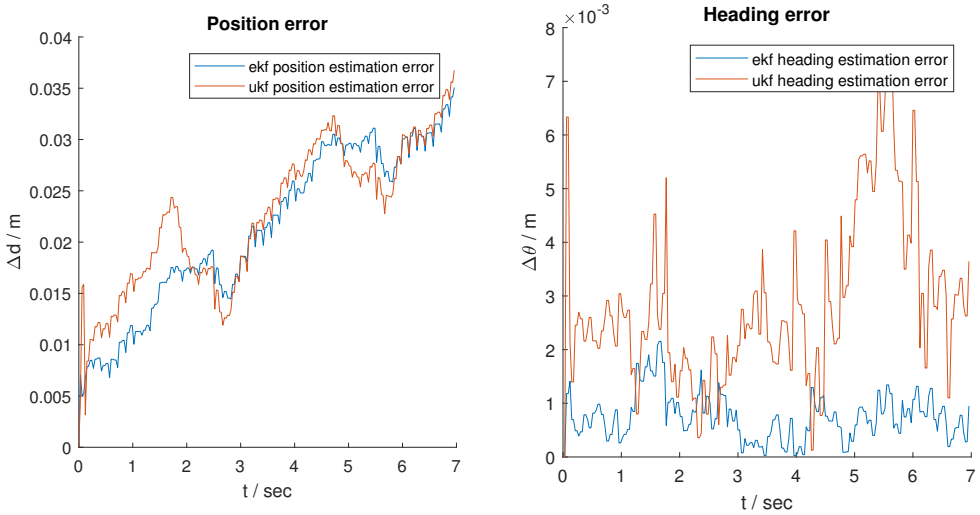


Abbildung 6.2: *Pfad des Roboters bei einer geradlinigen Bewegung entlang des Flurs bei dynamisch angelegten Markern.*



(a) *Schätzfehler der Position.*

(b) *Schätzfehler der Orientierung.*

Abbildung 6.3: *Vergleich der Lokalisierungsfehler des Roboters vom EKF und UKF bei dynamisch angelegten Markern.*

mittleren Abweichungen wurden in der Tabelle 6.1 dargestellt. Man erkennt, dass der Fehler in der gleichen Größenordnung liegt wie die Roboterpose. Die Tendenz ist, dass der Markerfehler größer als die geschätzte Markerpose ist. Diese Messung entspricht der Erwartung, dass bei vergleichbar wenigen Korrekturmöglichkeiten für einen einzelnen Marker der Schätzfehler größer ist als die Roboterpose, die bei jeder Messung korrigiert wird.

	Positionsfehler	Orientierungsfehler
EKF	bis zu 0.0453 m	bis zu 0.0065 rad
UKF	bis zu 0.0485 m	bis zu 0.0069 rad

Tabelle 6.1: *Mittlerer Trackingfehler der erkannten Marker.*

6.1.2 Markerlokalisierung auf der statischen Karte

Im nächsten Versuch wurde die gleiche geradlinige Bewegung durchgeführt, mit dem Unterschied, dass die Markerposen exakt bekannt waren und bei Lokalisierung verwendet wurden. Abb. 6.4 zeigt den resultierenden Pfad. Man erkennt nun keine bleibende Abweichung von der Originaltrajektorie.

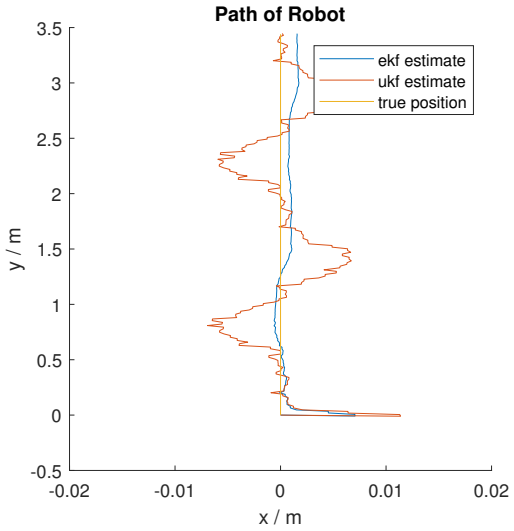
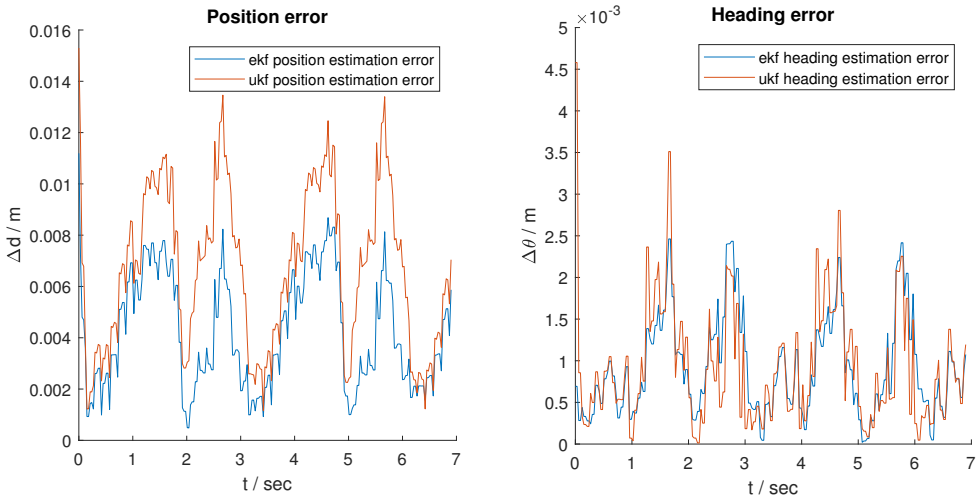


Abbildung 6.4: *Pfad des Roboters bei einer geradlinigen Bewegung entlang des Flurs bei statisch bekannten Markern.*

In Abb. 6.5 erkennt man keine wachsende Tendenz der Positions-/Orientierungs-abweichung. Eine obere Schranke für die Positionsabweichung war bei 0.009 m beim EKF (UKF: 0.014 m) zu erkennen. Der maximale Orientierungsfehler lag für den EKF bei ca. 0.15 deg (UKF: ca. 0.2 deg).

In beiden Diagrammen aus Abb. 6.3 und 6.5 ist ersichtlich, dass der Fehler beim EKF geringer ausfällt als beim UKF. Dennoch zeigen die Diagramme ebenfalls, dass die Schwankung der Fehlerwerte beim UKF größer als beim EKF war. Dieser Effekt konnte beim UKF durch einen charakteristischen Parameter[2] eingestellt werden. Daher hat der

[2] In [WM00] wird der Parameter für die Streuung als α definiert und wurde hier zu 1e-3 gesetzt



(a) Schätzfehler der Position. *(b) Schätzfehler der Orientierung.*

Abbildung 6.5: *Vergleich der Lokalisierungsfehler vom EKF und UKF bei dynamisch angelegten Markern.*

UKF trotz seiner geringeren Schätzgenauigkeit bei sowohl dynamischer und statischer Markerkarte eine größere Flexibilität bzgl. seiner Einstellmöglichkeiten, d.h. ein ähnliches Ergebnis zum EKF ist durch eine geeignete Wahl der Filterparameter bei gleichem $\mathbf{Q}$ und $\mathbf{R}$ möglich.

6.2 Untersuchungen zum Kollisionsvermeidungssystem

Um das Kollisionsvermeidungssystem zu testen, wurden zwei Versuchsreihen entworfen, die jeweils die Reaktion des Roboter gegenüber statischen Objekten und dynamischen Objekten analysiert. Das Ziel der Versuchsreihe ist die Überwindung einer vorgegebener Strecke (hier der Flur des Instituts) , wobei sich Hindernisse auf dem Weg befinden können. Die Priorität bei diesen Tests lag dabei auf das Stoppen der Bewegung bei kritischen Situationen. Dies beinhaltet u.a. plötzlich auftretende Objekte oder nicht passierbare Passagen. Ein weiterer Aspekt war das Fahrverhalten unter Berücksichtigung der non-holonomic constraints und des rechteckigen Roboterumrisses. Die Ausdehnungsparameter des Roboters wurde im Kollisionsvermeidungsmodul zu $w_{robot} = 0.42$ und $l_{robot} = 0.6$ gesetzt, d.h. der Roboter ist 0.84 m breit und 1.2 m lang. Die Parameter der Gewichtungsfunktion sind $\alpha = 0.2, \beta = 0.2, \gamma = 0.2$.

6.2.1 Kollisionsvermeidung von statischen Objekten

Die erste Versuchsreihe betrachtete die statischen Hindernisse. Dabei waren diese so angeordnet, dass die Durchfahrt enger war als die Länge des Roboters, d.h. die Annäherung als kreisförmigen Umriss war nicht ausreichend. Der Aufbau eines Versuchs wird beispielhaft in Abb. 6.6 dargestellt. Dabei symbolisiert die blaue Fläche den Roboter und die gelben Flächen die Hindernisse. x_{pass} ist der Versatz der Durchfahrt auf der x-Achse im Weltkoordinatensystem und d_{pass} ist die Breite der Durchfahrt. Die genauen Parameter in einem bestimmten Versuch können aus Tabelle 6.2 entnommen werden.

Versuch	Passagenversatz x_{pass}	Durchfahrtsbreite d_{pass}
1	0.0 m	1.1 m
2	0.05 m	1.0 m
3	0.1 m	0.9 m
4	-0.05 m	1.0 m
5	0.0 m	0.9 m
6	0.05 m	0.8 m
7	-0.1 m	0.9 m
8	-0.05 m	0.8 m
9	0.0 m	0.7 m

Tabelle 6.2: *Charakteristische Parameter der Versuchreihe mit statischen Objekten.*

Bei Durchfahrten mit $d_{pass} < 2 \cdot w_{robot}$ stoppt der Roboter vor dem Hindernis in der spezifizierten Toleranzgrenze ϵ_{front}. Es war erkennbar, dass der Roboter bei bestimmten Konfigurationen Probleme bei der Durchfahrt zwischen den Boxen hat. Bei einem versetzten Gang war ein Manöver notwendig um sich zunächst auszurichten. Dadurch waren enge Durchfahrten allein mit der Kollisionsvermeidung nicht mehr möglich, obwohl die Ausdehnung des Roboters es zulässt. Bei diesen Situationen blieb der Seekur Jr im erwarteten Bereich vor dem Hindernis stehen.

Außerdem wurde beobachtet, dass die Integration der Punkte aus der metrischen Karte sowohl Vorteile, als auch Nachteile für die Kollisionsvermeidung hatte. Man erkennt zum Beispiel in Abb. 6.7a, dass die kartierten Punkte aus der metrischen Karte dazu verwendet wurden Distanzen zu Objekten zu bestimmen, die nicht mehr im Sichtbereich des Lidars waren. In Abb. 6.7b hingegen sieht man, dass die Durchfahrt zwischen dem Objekt verhindert wurde aufgrund von falsch kartierten Punkten. U.a. waren die Objekte dem Zellraster nicht perfekt ausgerichtet, sodass die Auflösung der Rasterkarte ein Einflussfaktor war.

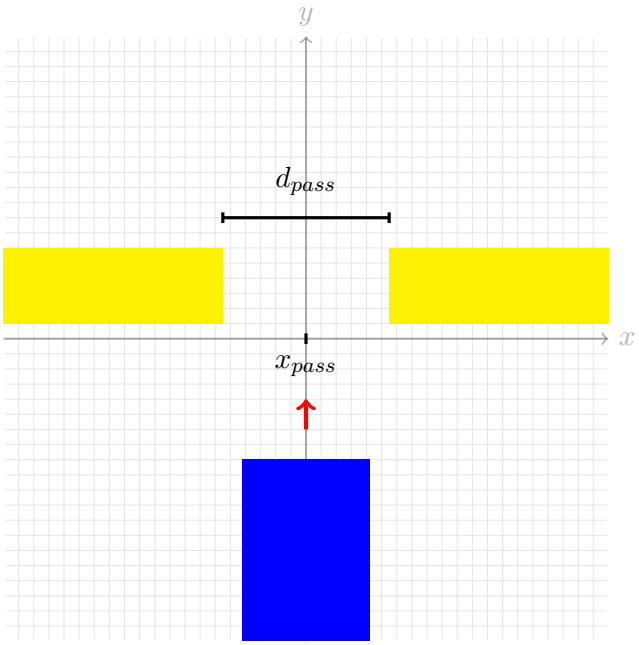


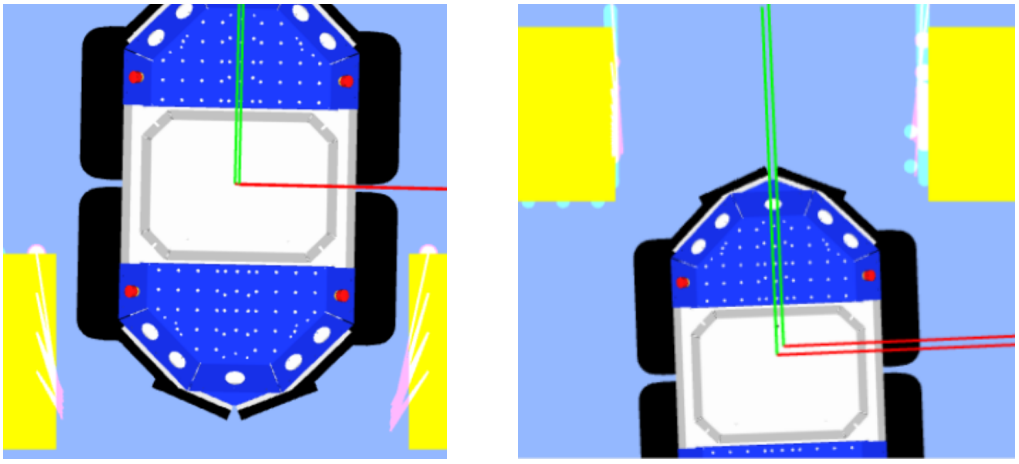
Abbildung 6.6: *Schematischer Aufbau der Versuchreihe mit statischen Objekten.*

Außerdem wurde ein Effekt im Fahrverhalten beobachtet, die auf die kartierten Punkte in $\mathbf{p}_{coll}$ zurückzuschließen waren. Bei einigen Versuchen dieser Versuchsreihe mit $x_{pass} \neq 0$ trat der Fall ein, dass der Roboter „Hindernisse in seiner Rotationsachse behalten wollte“, d.h. er umkreiste Objekte, an denen er nah vorbeifuhr bis eine Winkelabweichung zum Ziel erreicht wurde, die von dem Abstand zum Objekt abhängig war.

Ein Erklärungsansatz ist die Rotationsvarianz der Pose, welcher sich auf die Distanz der Kollisionspunkte auswirkt. Dadurch ist die Gewichtung der Pose von der Orientierung des Roboters abhängig. Da die Posenkonfigurationen im nächsten Schritt durch das Bewegungsmodell eingeschränkt sind, ist die Wahl der optimalen Kontrollvariablen von den relativen Gewichtungen zwischen der Gewichtung der Ausrichtung zum zielführenden Pfad ($\alpha \cdot heading(u_v, u_\omega)$) und der Gewichtung der Vermeidung von Objekten ($\beta \cdot clear(u_v, u_\omega)$) abhängig. Ein möglicher Lösungsansatz wäre eine vom Umriss unabhägige Definition von $clear(u_v, u_\omega)$, welche rotationsinvariant ist.

6.2.2 Kollisionsvermeidung von dynamischen Objekten

Die zweite Versuchsreihe beschäftigte sich mit der Reaktion auf dynamische Objekte. Hierbei bewegte sich ein Objekt in den Pfad des Roboters. Das Szenario wurde so gestaltet, dass der Roboter entlang des Flurs fuhr und von der Seite ein Hindernis mit einer konstanten Geschwindigkeit von $1\frac{m}{sec}$ sich vor den Roboter bewegte. Der Aufbau ist in



(a) Visualisierung der Kollisionsdistanz zu Hindernispunkten aus der Rasterkarte. *(b) Fehlerkennung von Hindernissen durch falsch kartierte Punkte.*

Abbildung 6.7: *Detaillierte Betrachtung der Rasterkartenpunkte in der Kollisionsvermeidung.*

Abb. 6.8 dargestellt, wobei die roten Pfeile die Bewegungsrichtungen der Roboters bzw. des Hindernisses darstellen.

In diesem Versuch wurde beobachtet, dass die Reaktion des Roboters im Wesentlichen von der Rate abhängt, mit der die Sollvariable generiert wird. In Alg. 4.1 wurde sie als Δt bezeichnet und bestimmt die Größe des Dynamic Windows und somit die Länge des Pfades, die „vorausgeplant“ wird. Bei hohen Raten wird für einen kurzen Zeitabschnitt ein Geschwindigkeitspaar generiert, der die momentanen Distanzmessungen berücksichtigt. Solange eine Bremsung innerhalb dieser Zeitperiode möglich ist, kann auch der Roboter darauf reagieren. In anderen Worten werden dynamische Objekte jeweils statisch in den „Frames“ wahrgenommen. Diese Theorie deckt sich auch mit der Beobachtung dieses Versuchs. Dennoch ist die Zeitperiode durch die Rechenzeit des DWA-Moduls limitiert. Bespielsweise wurde bei diesem Versuch eine Stichprobe der Ausführzeit[3] von ca. 100 ms im Visual-Studio-Debugger gemessen. Obwohl diese Stichprobe nicht repräsentativ für die Performance des Moduls ist, kann man die obere Grenze der maximalen Frequenz in den Bereich um 10 Hz einordnen.

[3] Rechenzeit des DWA-Moduls von der Verarbeitung der anliegenden Sensordaten bis zur Ausgabe des optimalen Geschwindigkeitpaars

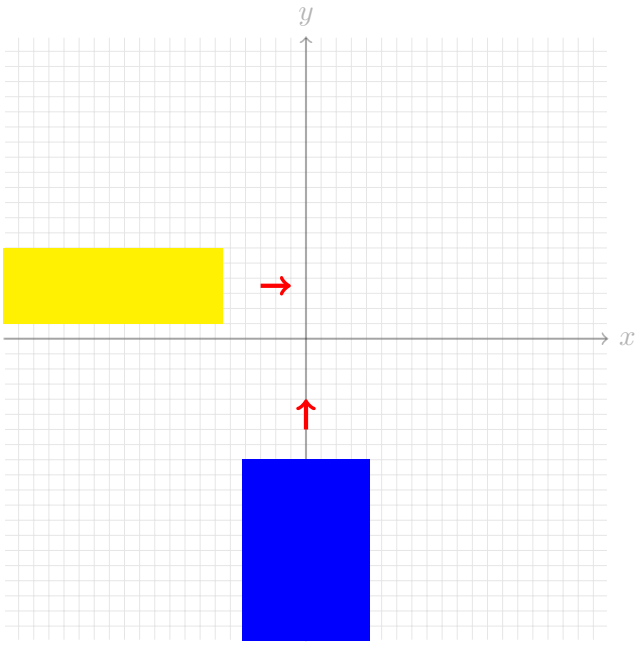


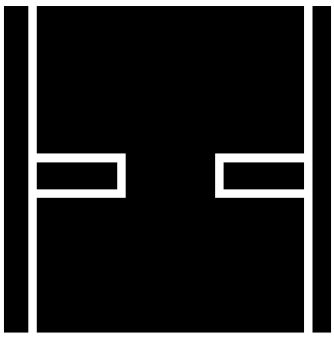
Abbildung 6.8: *Schematischer Aufbau der Versuchreihe mit dynamischen Objekten.*

6.3 Qualität der Rasterkarte

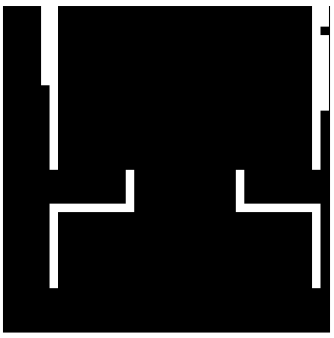
Um die Genauigkeit der Rasterkarte zu untersuchen, wurde der folgende Versuch durchgeführt: Es wurde für diesen Versuch die Umgebung aus Szenario 1 der Tab. 6.2 verwendet. Der Roboter fuhr zwischen den Boxen durch und generiert mit den aufgenommenen Laserdistanzen die metrische Karte mit dem in 3.3 vorgestellten Algorithmus 3.3. Dabei wurde die Karte in der näheren Umgebung der Boxen genauer betrachtet. Für den Vergleich werden die entsprechenden binären Rasterkarten verwendet. Das Thresholding wurde mit $p_{thresh} = 0.75$ durchgeführt. Die verwendeten Parameter des inversen Sensormodells sind $\delta_1 = 0.05$, $\delta_2 = 0.15$, $\delta_3 = 0.2$. Man beachte, dass die hier gewählten Parameter willkürlich gewählt worden sind, mit der Ausnahme, dass die Breite $\delta_2 - \delta_1$ der Zellgröße entspricht. In diesem Beispiel ist der Ursprung des Kartenkoordinatensystem zum Weltkoordinatensystem um $(-20, 20)^T$ verschoben und um 180 deg gedreht[4]. Die Indizierung der Zellen im Kartenkoordinatensystem erfolgt durch $(n, m) = (\lfloor \frac{x}{res} \rfloor, \lfloor \frac{y}{res} \rfloor)$, wobei $res = 0.1m$ die Auflösung der Karte ist.

Vergleicht man zunächst die erwartete Ground Truth Karte (Abb. 6.9a) mit der Karte (Abb. 6.9b), die aus der exakter Roboterpose bzw. Sensorpose generiert wird, dann erkennt man einen Offset der belegten Zellen, der eine Zelle in x- und y-Richtung des Kartenkoordinatensystems beträgt. Da das Zentrum des Flurs aufgrund der Indexkon-

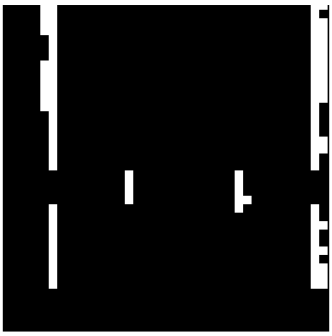
[4] in der 2D-Repräsentation der Welt



(a) Ground Truth Karte.



(b) Mapping mit exakter Pose.

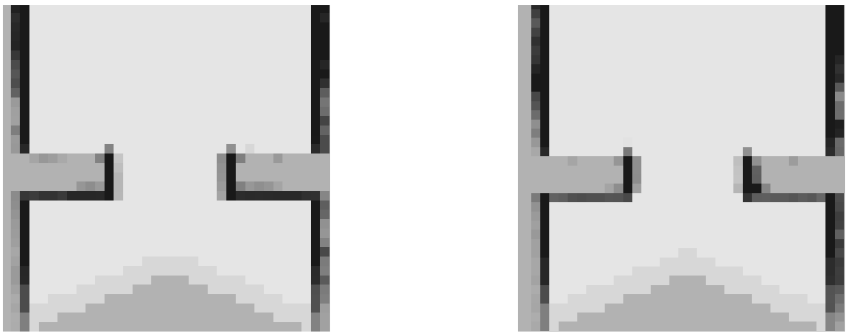


(c) Mapping mit geschätzter Pose (EKF).

Abbildung 6.9: *Vergleich der metrischen Kartengeneration in Form der binären Rasterkarte (weiß: belegt, schwarz: frei).*

vention auf dem Rand zwischen zwei Zellen liegt und damit auch der Pfad des Roboters, können Verschiebungen der Kartierung trotz exakter Pose auftreten, die sich im Bereich von einer Zellgröße befinden. Außerdem erkennt man unterschiedliche Hindernisbreiten, die durch die Parameter $\delta_1, \delta_2, \delta_3$ beeinflussbar sind. Man erkennt aus Abb. 6.9b ebenfalls, dass die hintere Grenze der Box bei der Durchfahrt nicht erkannt wurde. Dies entspricht der Erwartung, dass bei dem limitierten Öffnungswinkel des Lasersensors (siehe grüne Fläche aus Abb. 4.2a) entweder die Grenze als Messung gar nicht oder nur für eine kurze Zeit vorhanden war, sodass der Wahrscheinlichkeitswert nicht über den Schwellwert p_{thresh} konvergierte.

Der Unterschied zwischen der Kartierung mit der geschätzten Pose in Abb. 6.9c und der genauen Pose (Abb. 6.9b) ist die größere Unsicherheit. Es ist ersichtlich, dass sich die erkannten Objekte in der gleichen Position befinden, aber die Wahrscheinlichkeiten anders verteilt sind. In Abb. 6.10 kann man den Unterschied mittels der Grauwerten erkennen. Dies wirkte sich nach dem Thresholding auf die binäre Rasterkarte aus, indem die vordere Kante der Box als nicht belegt markiert wurde.



(a) *Mapping mit exakter Pose.* ***(b)*** *Mapping mit geschätzter Pose (EKF).*

Abbildung 6.10: *Vergleich der Rasterkarte mit probabilistischen Werten.*

Dieses Ergebnis deckt sich mit den Erwartungen, dass eine genauere Kartierung mit einer genauen Posenschätzung möglich ist. Dennoch wurden hier die Information der metrischen Karte nicht für die Lokalisierung verwendet, sodass die Kartierungsgenauigkeit ausschließlich von der Genauigkeit der Posenschätzung abhängig ist.

6.4 Qualitative Auswertung des Feldtest im MMI Flur

Im Vergleich zu der Simulation sind die Daten in der realen Testumgebung nicht idealisiert und stark rauschbehaftet. Ein Versuch mit dem in Kap. 5.4 beschriebenen Prototypen wurde durchgeführt, um Unterschiede zwischen Simulation und realem Testaufbau zu beobachten. Dieser war wie folgt spezifiziert: Der Roboter sollte mit der EKF-Lokalisierung 10 m den Flur entlang fahren und auf statische / dynamische im Weg reagieren. Die Marker waren entlang des Flurs in regelmäßigen Abständen (ca. 3 - 5 m Abstand) verteilt in einer Höhe[5] von ca. 56.5cm . Die Flurbreite dabei betrug ca. 3.65 m. Die Kantenlänge war vorgegeben mit 28.9 cm. Die exakten Markerposen sind in Tab. B.4 gegeben.

[5] vom Boden zur Markermitte

Neben der Fehlerkennung von Objekten mit dem Laserscanner wie z.B. von Glasscheiben und schwarzen Jeans/Anzughosen, wurden Lokalisierungssprünge bis zu 1-2m beobachtet. Dennoch war die Lageschätzung des Roboters im längeren Verlauf des Versuchs tendenziell genauer als das Dead-Reckoning-Verfahren. Außerdem zeigte das System bei max. 50% der geplanten Geschwindigkeit ein robustes Verhalten in der Fahrdynamik, d.h. sicheres Ausweichen von Objekten und Notabschaltung bei Hindernissen in der kritischen Zone. Die Rasterkarte wies schnell Inkonsistenzen auf. Dies konnte man mit dem Fehler in der Lokalisierung begründen.

Obwohl diese Arbeit keine genauen Performanceuntersuchungen der Module vornimmt, war es offensichtlich, dass die Echtzeitfähigkeit eine zentrale Rolle im realen Prototypen einnahmen. Z.B. konnten die Instabilitäten des Systems bei höheren Geschwindigkeiten (> 50 %) durch Verarbeitungsengpässe in den Modulen begründet werden. Das Problem mit den Lokalisierungssprüngen können letztendlich mit einer Verbesserung der Modulperformance gelöst werden. Eine Erklärung für die Lokalisierungssprünge wäre, dass die Lokalisierung in bestimmten Abschnitten der Trajektorie ausschließlich mit der Odometrie erfolgt und somit die Unsicherheit der Pose zunehmend ansteigt. Bei höheren Bildauflösungen (z.B. 1280x1024) können kontinuierlich valide Marker erkannt werden, sodass abwechselnd Prädiktionsschritte und Korrekturschritte erfolgen können.

KAPITEL 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Es wurden in dieser Arbeit Plugins in der VEROSIM Umgebung implementiert, die in ihrer Gesamtheit ein System zur Lokalisierung und Navigation von mobilen Robotern bilden. Dabei wurde sich auf die Modularität der Funktionen konzentriert, sodass diese unabhängig u.a. in anderen Anwendungsszenarien voneinander verwendet können.

Die implementierten Filteralgorithmen EKF und UKF wurden für den Marker-basierte SLAM Algorithmus verwendet. Dieser kann mithilfe hybrider Markerkarten, d.h. sowohl bekannten als auch unbekannten Markern, seine relative Pose im Raum schätzen. Die Schätzung auf bekannten Markern zeigte ein robustes Verhalten. Bei einer ausschließlich dynamisch generierten Markerkarte erkennt man einen ansteigenden Schätzfehler, welcher auf die SLAM Problematik zurückzuführen ist. Bei der Wahl der Standardparameter[1] für den UKF war der Schätzfehler des EKFs tendenziell geringer. Dennoch ist noch viel Spielraum bei der Einstellung der Filterparameter vorhanden. Die Schätzgenauigkeit ist ebenfalls stark von den Messdaten abhängig. Fehlmessungen werden im Filterverfahren nicht modelliert, sodass eine Vorfilterung der Messdaten notwendig ist. Zudem ist die Schätzung gegenüber Ausreißern in den Messdaten instabil.

Die Präzision der Rasterkarte ist ausschließlich von der Genauigkeit der Posenschätzung abhängig. D.h. bei einer ungenauen Lokalisierung werden die Hindernisse in der Rasterkarte ungenau kartiert. Da es kein Feedback von der Karte zurück zur Pose gibt, können Inkonsistenzen in der Karte auftreten.

Es wurde in dieser Arbeit ersichtlich, dass die Berücksichtigung des Roboterumrisses mehr Manöver erlaubt als bei einer generellen Annäherung durch eine Kreisform. Die lokale Pfadplanung zeigt aber einen suboptimalen Lösungsweg bei der dadurch entstehenden Rotationsinvarianz der Gewichtskomponente zum Vermeiden von Objekten. Die

[1] aus [WM00]

Integration der Rasterkartenzellen im DWA erweist sich als valide Option, um Bereiche abzudecken, die von der Sensorik der Kollisionsvermeidung nicht erfasst werden können. Dennoch schränken mögliche Kartierungsfehler den Bewegungsraum ein. Die Recheneffizienz des Kollisionsvermeidungsalgorithmus ist die Schlüsselkomponente zu einer schnellen reaktiven Planung, da die Umwelt quasi-statisch wahrgenommen wird. Das radial sampling wurde mit der Absicht einer effizienteren Berechnung von Komponenten implementiert. Eine detaillierte Analyse der Performance sowie Optimierungen in der Programmebene stehen noch aus.

7.2 Ausblick

Da die Module dieses Systems nach dem Top-down Ansatz für den Seekur Jr entwickelt wurden, sind zahlreiche Arbeiten möglich, um die Funktionalität der implementierten Plugins zu erweitern. Die folgenden Vorschläge beschränken sich auf mögliche Untersuchungen, um die vorgestellten Module zu optimieren und ein tieferes Verständnis für diese zu schaffen.

Neben der Parameterkonfiguration des UKFs kann bzgl. der Lokalisierung die Fusion des marker-basierten SLAM und dem SLAM mit der Rasterkarte untersucht werden. So können genauere und konsistente Gridmaps erzeugt werden. Da die Implementierung der Gridmap in dieser Arbeit nur für die Kompensation von „toten Winkeln“ verwendet wurde, ist hier viel Spielraum für Optimierungen.

Für die Navigation ist die Integration von globalen Planern sinnvoll, um einen optimalen Weg zum Ziel zu generieren. Davor sollte die Problematik der lokalen Planung mit rotationsvarianten Umrissen gelöst werden. Zum Beispiel können die hard constraints der Kollisionsvermeidung weiterhin mit dem exakten Umriss berechnet werden, wobei die Auswahl des optimalen Geschwindigkeitstupels über die soft constraints über eine rotationsinvariante Annäherung des Roboterumrisses erfolgt.

ANHANG A

Detaillierte Beschreibung verwendeter Unterfunktionen

A.1 Gewichtsdefinitionen beim Unscented Kalman Filter

Die Gewichte $w_c^{(i)}$ und $w_m^{(i)}$ sind für Zustände der Dimension N wie folgt definiert [WM00]:

$$w_m^{(i)} = \begin{cases} \frac{\lambda}{N+\lambda} & i = 0 \\ \frac{\lambda}{2(N+\lambda)} & i = 1, \ldots, 2N \end{cases} \tag{A.1}$$

$$w_c^{(0)} = \begin{cases} \frac{\lambda}{N+\lambda} + (1 - \alpha_w^2 + \beta_w) & i = 0 \\ \frac{\lambda}{2(N+\lambda)} & i = 1, \ldots, 2N \end{cases} \tag{A.2}$$

Es gilt dabei $\lambda = \alpha_w^2(N + \kappa) - N$. In dieser Arbeit wurden $\alpha_w = 0.001$, $\beta_w = 2$ und $\kappa = 0$ als Parameter verwendet.

A.2 Gewichtsfunktionen und gesetzte Parameter des DWAs

Die Untergewichte der Objective Function sind als

$$head(u_v, u_\omega) = \frac{\pi - \varphi(u_v, u_\omega)}{\pi} \tag{A.3}$$

$$clear(u_v, u_\omega) = \min_i dist_{fp}(\mathbf{p}_{i,coll}) \tag{A.4}$$

$$vel(u_v, u_\omega) = \begin{cases} \frac{1-u_v}{v_{max}} & sonst \\ \frac{u_v}{v_{max}} & Zielumgebung \end{cases} \tag{A.5}$$

definiert. Es beschreibt φ den Winkel von der Orientierung des Roboters zum Ziel und $dist_{fp}(\mathbf{p}_{i,coll})$ die minimale Distanz vom Punkt zum Roboterumriss. Die Untergewichte sind jeweils gleichgewichtet, d.h. $\sigma = 1$ und $\alpha = \beta = \gamma = 0.2$.

ANHANG B

Detaillierte Beschreibung verwendeter Parameter

B.1 Simulationsparameter in VEROSIM

Simulationszeitschritt:	30 ms
Bildrenderperiode:	30 ms
Timer Mode - Ratio Cap:	0

Tabelle B.1: *Generelle Simulationsparameter.*

Zeitschritt:	10 ms
Kontaktmodell:	Spring Damper

Tabelle B.2: *Simulationsparameter der Dynamiksimulation.*

B.2 Markerposen in der Flursimulation

ID	x	y	θ
0	-1.499	0.9	0.0
1	1.499	1.1	3.141592
2	-1.499	3.9	0.0
3	1.499	4.1	3.141592
4	1.499	-1.9	3.141592
5	-1.499	-2.1	0.0
6	-1.499	-0.6	0.0
7	1.499	-0.4	3.141592
8	1.499	2.6	3.141592

9	-1.499	2.4	0.0
10	-1.499	5.4	0.0
11	1.499	5.6	3.141592

Tabelle B.3: *Posen der statischen Markerkarte in der Flursimulation.*

B.3 Markerposen im MMI-Flur

ID	x	y	θ
0	-1.825	1.231	0.0
5	-1.825	5.643	0.0
14	-1.825	11.249	0.0
17	-1.825	15.238	0.0
20	-1.825	19.232	0.0
26	-1.825	23.235	0.0
34	1.825	3.393	3.141592
39	1.825	7.996	3.141592
41	1.825	12.900	3.141592
44	1.825	16.887	3.141592
63	1.825	21.285	3.141592
101	-2.625	9.306	3.141592
102	-2.950	10.938	-1.5708
103	-1.825	28.350	0.0
104	1.825	28.350	3.141592
105	-1.825	32.350	0.0
106	1.825	32.850	3.141592
107	-1.825	36.350	0.0
108	1.825	36.350	3.141592

Tabelle B.4: *Posen der statischen Markerkarte im MMI-Flur.*

B.4 Parameter der Kamerakalibrierung

RMS	camera matrix	distortion coefficients
0.5064	$\begin{pmatrix} 1377 & 0 & 783 \\ 0 & 1376 & 467 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0.06676 \\ -0.6463 \\ -0.0008233 \\ -0.0006451 \\ 1.0239 \end{pmatrix}$

Tabelle B.5: *Kameraparameter der Logitech HD Webcam C510 (in OpenCV Konvention).*

Literaturverzeichnis

[AG17] S. AG: *SICK LMS151 Datasheet*. 2017.

[APT+02] K. Arras, J. Persson, N. Tomatis et al.: *Real-time obstacle avoidance for polygonal robots with a reduced dynamic window*. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. IEEE, 2002. DOI: 10.1109/robot.2002.1013695.

[BK99] O. Brock und O. Khatib: *High-speed navigation using the global dynamic window approach*. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. IEEE, 1999. DOI: 10.1109/robot.1999.770002.

[Cor96] P. Corke: *A robotics toolbox for MATLAB*. In: *IEEE Robotics & Automation Magazine* 3.1 (1996), S. 24–32. DOI: 10.1109/100.486658.

[D G16] V. M. D. Gonzales J. Perez: *A Review of Motion Planning Techniques for Automated Vehicles*. In: *IEEE Transactions on Intelligent Transportation Systems Vol.17 No. 4* (2016).

[Die97] S. T. Dieter Fox Wolfram Burgard: *The Dynamic Window Approach to Collision Avoidance*. In: *IEEE Robotics and Automation Magazine* (1997).

[Doc18] OpenCV: *Tutorial ArUco Detection*. 2018. URL: https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html (besucht am 19. 03. 2018).

[FBT96] D. Fox, W. Burgard und S. Thrun: *Controlling synchro-drive robots with the dynamic window approach to collision avoidance*. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*. IEEE, 1996. DOI: 10.1109/iros.1996.568982.

[G G05] C. S. u. W. B. G. Grisetti: *Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling*. In: *IEEE International Conference on Robotics an Automation* (2005).

[GMM+14] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas et al.: *Automatic generation and detection of highly reliable fiducial markers under occlusion*. In: *Pattern Recognition* 47.6 (2014), S. 2280–2292. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/j.patcog.2014.01.005.

[IPA18] F. IPA: *Roboter als vielseitiger Gentleman*. 2018. URL: https://www.ipa.fraunhofer.de/de/presse/presseinformationen/2015-01-15_roboter-als-vielseitiger-gentleman.html (besucht am 02. 03. 2018).

[KG09] P. Kabamba und A. Girard: *Fundamentals of Aerospace Navigation and Guidance*. Cambridge University Press, 2009. doi: 10.1017/cbo9781107741751.

[Kha85] O. Khatib: *Real-time obstacle avoidance for robot manipulator and mobile robots*. In: *The International Journal of Robotics Research* (1985).

[LLC18] O. A. M. LLC: *ARIA documentation and manual*. 2018. url: http://robots.mobilerobots.com/wiki/ARIA.

[Log10] Logitech: *Logitech HD Webcam C510*. 2010.

[Lu14] D. V. Lu: *Contextualized Robot Navigation*. Diss. Washington University, St. Louis, 2014.

[Med18] A. MediaCenter: *Auszeichnung für Audi R8-Manufaktur*. 2018. url: https://www.audi-mediacenter.com/de/pressemitteilungen/auszeichnung-fuer-audi-r8-manufaktur-5569 (besucht am 02. 03. 2018).

[MM09] J. Minguez und L. Montano: *Extending Collision Avoidance Methods to Consider the Vehicle Shape, Kinematics, and Dynamics of a Mobile Robot*. In: *IEEE Transactions on Robotics* 25.2 (2009), S. 367–381. doi: 10.1109/tro.2009.2011526.

[Mor89] H. P. Moravec: *Sensor Fusion in Certainty Grids for Mobile Robots*. In: *Sensor Devices and Systems for Robotics*. Springer Berlin Heidelberg, 1989, S. 253–276. doi: 10.1007/978-3-642-74567-6_19.

[MSK+13] A. Maroti, D. Szaloki, D. Kiss et al.: *Investigation of Dynamic Window based navigation algorithms on a real robot*. In: *2013 IEEE 11th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 2013. doi: 10.1109/sami.2013.6480952.

[MSW02] M. Montemerlo, D. K. S. Thrun und B. Wegbreit: *FastSLAM, A Factored Solution to the Simultaneous Localization and Mapping Problem*. In: *AAAI* (2002).

[PS03] R. Philippsen und R. Siegwart: *Smooth and efficient obstacle avoidance for a tour guide robot*. In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. IEEE, 2003. doi: 10.1109/robot.2003.1241635.

[Rav15] P. K. Raveendran: *Markerbasierte Indoor-Lokalisierung*. Institut für Mensch-Maschine-Interaktion, RWTH, 2015.

[S T05] D. F. S. Thrun W. Burgard: *Probabilistic Robotics*. MIT Press, 2005.

[Sha99] B. Shamah: *Experimental Comparison of Skid Steering Vs. Explicit Steering for a Wheeled Mobile Robot*. Magisterarb. The Robotics Institute, Carnegie Mellon University, 1999.

[Sta16] J. M. Stanislaw Konatowski Piotr Kaniewski: *Comparison of Estimation Accuracy of EKF, UKF and PF Filters*. In: *The Journal of Polish Navigational Forum* (2016).

[TSM14] H. Tanaka, Y. Sumi und Y. Matsumoto: *A solution to pose ambiguity of visual markers using Moir-patterns*. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014. DOI: 10.1109/iros.2014.6942995.

[VER18] VEROSIM: *VEROSIM 1.2 Developer Documentation*. 2018.

[WM00] E. Wan und R. V. D. Merwe: *The unscented Kalman filter for nonlinear estimation*. In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*. IEEE, 2000. DOI: 10.1109/asspcc.2000.882463.

Eidesstattliche Versicherung

Loh, Johnson Luo	318755
Name, Vorname	Matrikelnummer *(freiwillige Angabe)*

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Masterarbeit mit dem Titel

SLAM-basierte Autonome Indoor-Navigation für mobile Roboter

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, 07. Mai. 2018	
Ort, Datum	Unterschrift

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen, 07. Mai. 2018	
Ort, Datum	Unterschrift