

# LOKALISATION MOBILER ROBOTER MIT ODOMETRIE UND BILDVERARBEITUNG IN EINER THEATERINSTALLATION

## M A S T E R A R B E I T

eingereicht am

23.09.2013

bei

Prof. Dr.-Ing. Udo Frese

Universität Bremen

von

Josef F. Hiller

Matr. Nr: 2055491

Osterstr 79

28199 Bremen



# Zusammenfassung

**Schlagwörter:** abc, def, xyz



# Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig angefertigt und mich fremder Hilfe nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichtem oder unveröffentlichtem Schrifttum entnommen sind, habe ich als solche kenntlich gemacht.

Bremen, 23.09.2013

---

Josef F. Hiller



# Inhaltsverzeichnis

<b>Zusammenfassung</b>	<b>iii</b>
<b>Erklärung</b>	<b>v</b>
<b>Abbildungsverzeichnis</b>	<b>ix</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Ziel und Aufgabenstellung der Arbeit . . . . .	1
1.2 Gliederung . . . . .	3
<b>2 Grundlagen</b>	<b>5</b>
2.1 Der Roboter . . . . .	5
2.2 Bühneninstallation . . . . .	6
2.3 Lokalisation . . . . .	6
2.3.1 Übersicht gängiger Verfahren . . . . .	8
2.3.2 Der Partikelfilter . . . . .	8
2.4 Bildverarbeitung . . . . .	11
<b>3 Lokalisierung mittels Bildverarbeitung</b>	<b>13</b>
3.1 Bitmuster . . . . .	13
3.2 Partikelfilter . . . . .	14
3.2.1 Zustandsraum . . . . .	14
3.2.2 Dynamikmodell . . . . .	15
3.2.3 Messmodell . . . . .	16
3.2.4 Initialisierung . . . . .	18
3.2.5 Schätzung aus Partikeln berechnen . . . . .	19
3.3 Musterbewertung . . . . .	19
3.3.1 Spezialfall: Projektion ergibt keine Bildpunkte . . . . .	19
3.3.2 Mindestanzahl Auswertbarer Bildpunkte . . . . .	22
3.3.3 Bestimmung mittlerer Weiß- und Schwarzwerte . . . . .	22

---

3.3.4	Bewertungskriterium . . . . .	22
3.4	Parameter einstellen . . . . .	23
<b>4</b>	<b>Die Simulationssoftware</b>	<b>27</b>
4.1	Koordinatensysteme . . . . .	27
4.2	Die Szene . . . . .	28
4.3	Messen in der Simulation . . . . .	30
4.3.1	Messwerte der Encoder . . . . .	30
4.3.2	Messabweichung der Encoder . . . . .	31
4.3.3	Bilder der Kamera . . . . .	32
<b>5</b>	<b>Versuche in der Simulation</b>	<b>35</b>
5.1	Versuche zur Genauigkeit des Messmodells . . . . .	35
5.2	Verwendete Fahrkurven . . . . .	38
5.3	Versuche zur Bildaufnahmefrequenz . . . . .	40
5.4	Versuche zum Einfluss von Menschen auf der Bühne . . . . .	42
<b>6</b>	<b>Fazit und Ausblick</b>	<b>45</b>
<b>A</b>	<b>Abkürzungsverzeichnis</b>	<b>I</b>
<b>B</b>	<b>Literaturverzeichnis</b>	<b>III</b>
<b>C</b>	<b>Anhang</b>	<b>V</b>



# Abbildungsverzeichnis

1.1	Roboter und Besucher auf der EPKOT . . . . .	2
1.2	Roboter vor Lichtwand auf der EPKOT . . . . .	2
2.1	Nahaufnahme eines der Roboter . . . . .	5
2.2	Bühnenmodell . . . . .	6
3.1	Bitmuster der Lichtwände . . . . .	13
3.2	1,8 Millionen Posen wurden mit dem Bild oben Bewertet. Nur 136 davon liegen über dem Minimalscore. Sie sind hier abgebildet. . . . .	21
4.1	Koordinatensysteme in der Simulation . . . . .	28
4.2	Simulierte Szene . . . . .	29
4.3	OpenGL frustum . . . . .	33
5.1	Antwort des Messmodells zu einem Bild an der rot markierten Posi- tion oben: 5cm Raster 3° Schritte unten: 1cm Raster 0.3° Schritte . . .	36
5.2	Antwort des Messmodells bei x=0.0 . . . . .	37
5.3	Fahrkurven oben gegen, und unten im Uhrzeigersinn . . . . .	39
5.4	Fahrkurven oben Schlangenlinien, unten schräge Ellipse . . . . .	41
5.5	Gemittelter Abstand von der Fahrkurve über verschiedene Bildauf- nahmefrequenzen mit vier verschiedenen Fahrkurven. . . . .	42
5.6	Mögliche Positionen von Personen . . . . .	43
5.7	toller text . . . . .	44
C.1	code 93 Tabelle . . . . .	VI



# 1 Einleitung

In dieser Arbeit soll, mit Hilfe einer Simulation, untersucht werden, wie gut sich Kameras bei der Standortbestimmung in mobilen Roboter einsetzen lassen. Anlass dieser Arbeit war eine Anfrage der Künstlergruppe *Beobachter der Bediener von Maschinen* (BBM), die schon bei mehreren Performances<sup>1</sup> mobile Roboter eingesetzt hat. Diese fahren zum Teil in einer Choreografie über die Bühne. Um sie dabei eine vorgegebene Figur fahren zu lassen, muss der Steuerung bekannt sein, wo sich ein Roboter auf der Bühne aufhält. Eine solche Lokalisation war in der Vergangenheit für BBM sehr aufwendig. Aus diesem Grund interessierte sie sich für eine Lokalisierungslösung welche möglichst ohne weitere Spezialhardware und geringem Installationsaufwand vor Ort auskommt. Der Ansatz der daraus entstand war: die Kameras, die bereits an jedem der Roboter verbaut waren, zu nutzen um markante Muster in der Bühneninstallation zu erkennen. Zusammen mit Messungen der Odometrie, sie beschreibt wie die Räder eines Roboters sich bei Bewegung drehen, soll die Position und Orientierung berechnet werden. Ein Partikelfilter wäre als Zustandsschätzer zu verwendet. Zu der Bühneninstallation gehören große Lichtwände, zu sehen auf Abbildung 1.1 und 1.2. Auf diesen Lichtwänden könnte ein hell/dunkel Bit-Muster angezeigt werden das es mit Hilfe geeigneter Bildverarbeitungsalgorithmen und mit den Kameras zu erkennen gilt.

## 1.1 Ziel und Aufgabenstellung der Arbeit

Im Rahmen dieser Arbeit soll eine Simulationsumgebung mit Hilfe geeigneter 3D-Visualisierungs Bibliotheken erstellt werden. Diese soll in der Länge sein eine 3D-Szene der Bühne zu simulieren und Kamerabilder sowie Odometrie-Daten einer Roboterfahrt zu erzeugen. Des weiten soll ein Lokalisationsalgorithmus entwickelt wer-

---

<sup>1</sup>unter Anderem:

2000 Themenpark "Wissen" der Expo 2000 Hannover

2010 Joybots in der BMW-Welt

2012 EPKOT Experimental Prototype Killers of Tomorrow , Hannover

siehe auch <http://www.bbm.cfnt3.de>

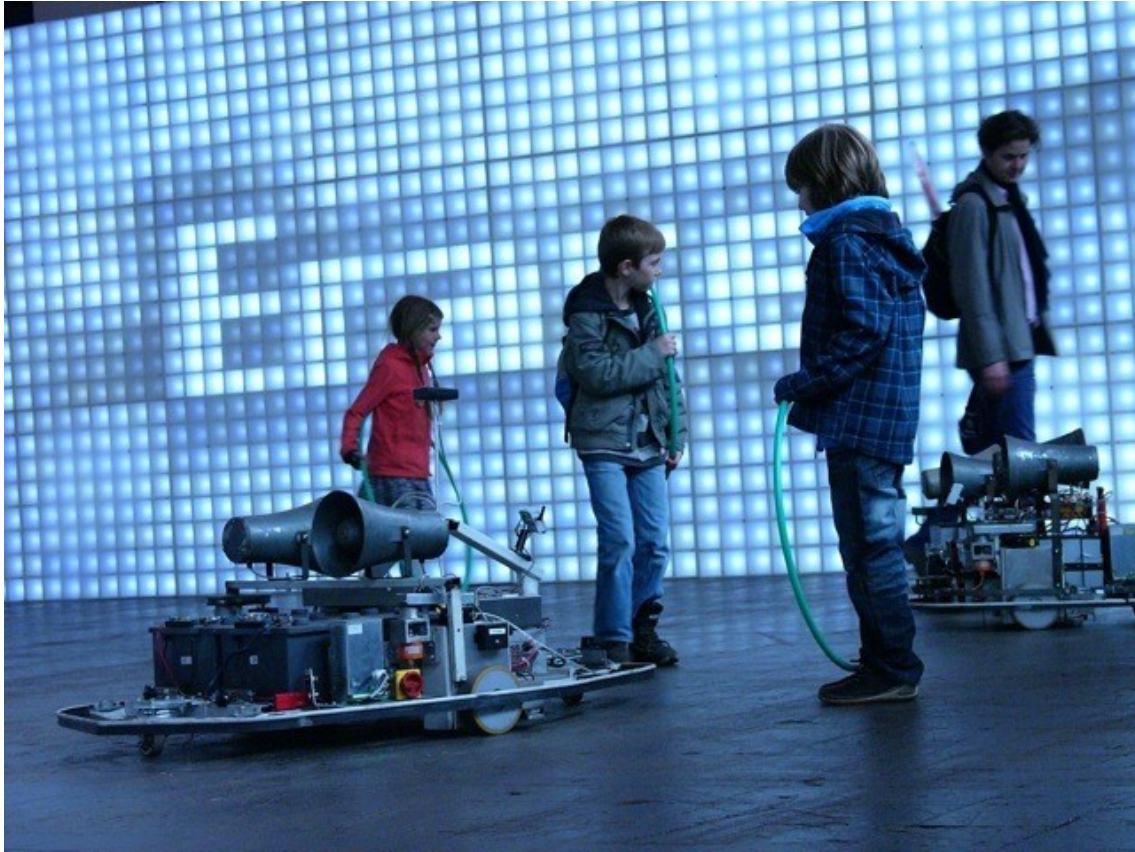


Abbildung 1.1: Roboter und Besucher auf der EPKOT Quelle: <http://www.bbm.cfnt3.de>

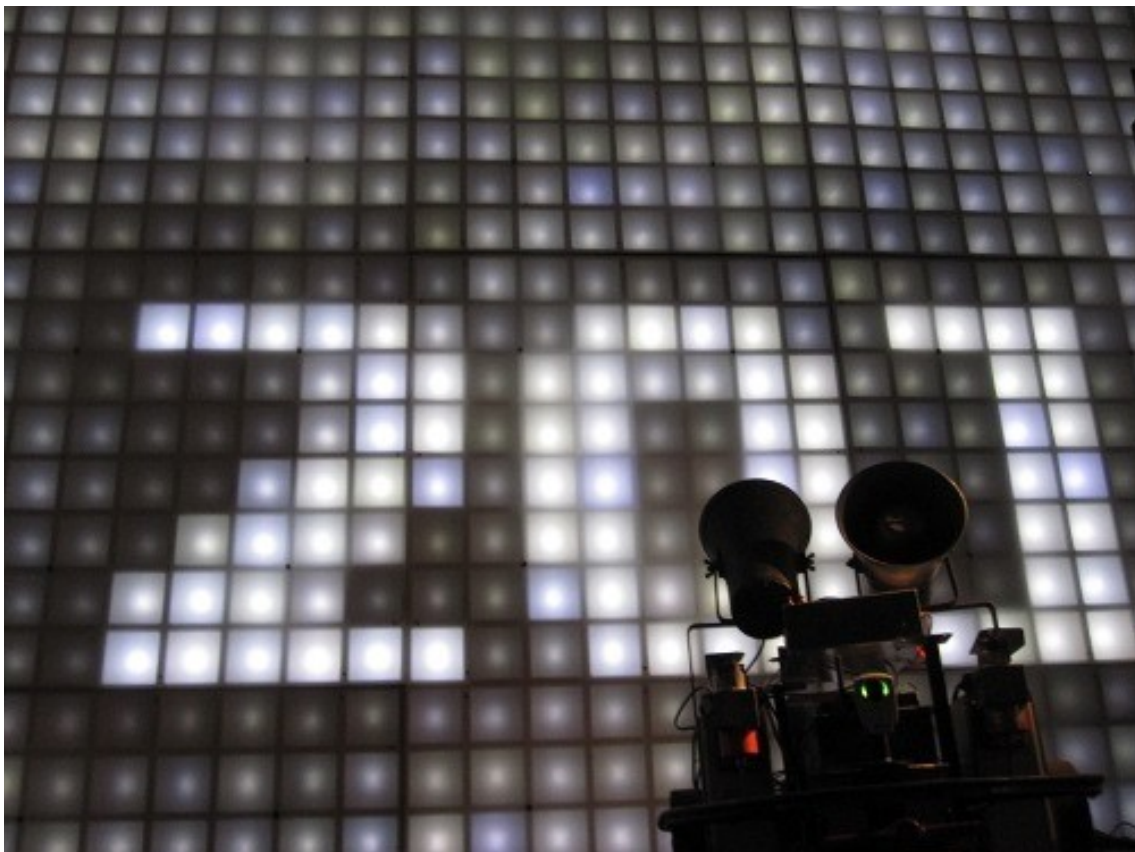


Abbildung 1.2: Roboter vor Lichtwand auf der EPKOT Quelle: <http://www.bbm.cfnt3.de>

den, der auf Grundlage dieser Daten die Position und Orientierung des Roboters auf der Bühne schätzen kann. Anschließend soll die Qualität dieser geschätzten Position beurteilt werden und mögliche Fehlerquellen diskutiert werden.

## 1.2 Gliederung

Die Arbeit wird im folgenden Kapitel eine kurze Übersicht gängiger Lokalisationsverfahren sowie deren Vor- und Nachteile geben. Außerdem wird in die Grundlagen eingeführt, welche zum Verständnis der folgenden Kapitel notwendig sind. Das Kapitel *Lokalisierung mittels Bildverarbeitung* beschreibt wie die Methoden aus den Grundlagen an das gestellte Problem angepasst wurden und wie das vorgestellte Verfahren funktioniert. Anschließend wird die Simulations- und Lokalisationssoftware vorgestellt, deren Implementation erklärt und begründet. Insbesondere wird darauf eingegangen, wie realistisch die Simulation ist. In Kapitel 5 beginnt die Beschreibung verschiedener Versuche die zum Beurteilen der Lokalisationsergebnisse durchgeführt wurden. Ergebnispräsentation und Diskussion erfolgen jeweils im Anschluss der Beschreibungen. Abschließend wird ein Ausblick zur möglichen Anwendung dieses Verfahrens gegeben, sowie mögliche Fehlerquellen und Probleme dabei. Im letzten Kapitel wird ein Fazit zu den Erkenntnissen dieser Arbeit gezogen.



## 2 Grundlagen

### 2.1 Der Roboter

An dieser Stelle soll kurz beschrieben werden, wie die Roboter aufgebaut sind die als Grundlage für die Simulation dienen. Auf den Abbildungen 1.1, 1.2 und 2.1 sind sie zu sehen. Sie besitzen zwei angetriebene Räder und mindestens ein weiteres Stützrad, welches frei drehbar ist. Die Räder können unabhängig voneinander angetrieben werden. Dazu sind pro Rad ein Servomotor mit Motorsteuerung und ein Getriebe mit Übersetzung verbaut. Zudem ist an jedem Rad ein Drehimpulsgeber angebracht, der

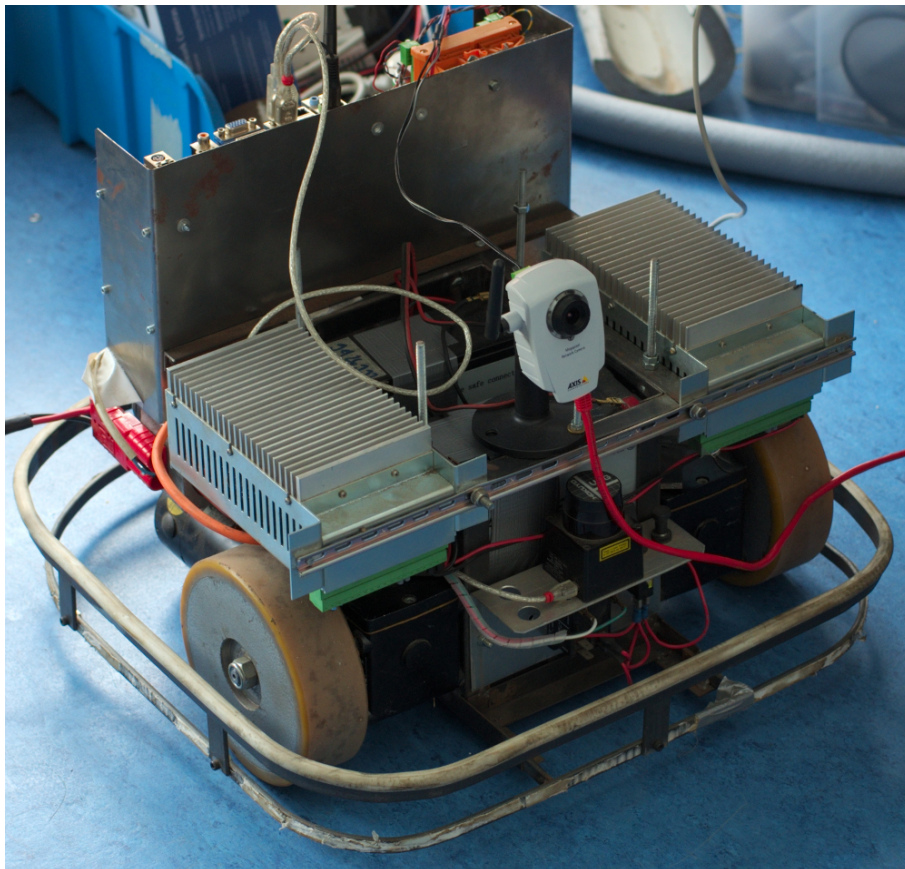


Abbildung 2.1: Einer der Roboter in Nahaufnahme

die Umdrehungen der Motorwelle misst. Dieser wird im weiteren Verlauf der Arbeit



mit Encoder<sup>1</sup> abgekürzt. Es gibt verschiedene Baugrößen der Roboter, der Antrieb ist aber bei allen baugleich. Die Räder haben einen Durchmesser von 160 mm, das Getriebe hat ein Übersetzungsverhältnis von 1:14,5 und die Encoder haben eine Auflösung von 2000 Ticks pro Umdrehung. Lediglich der Radstand unterscheidet sich bei den Baugrößen. Für die Simulation wird mit 700 mm gearbeitet.

## 2.2 Bühneninstallation

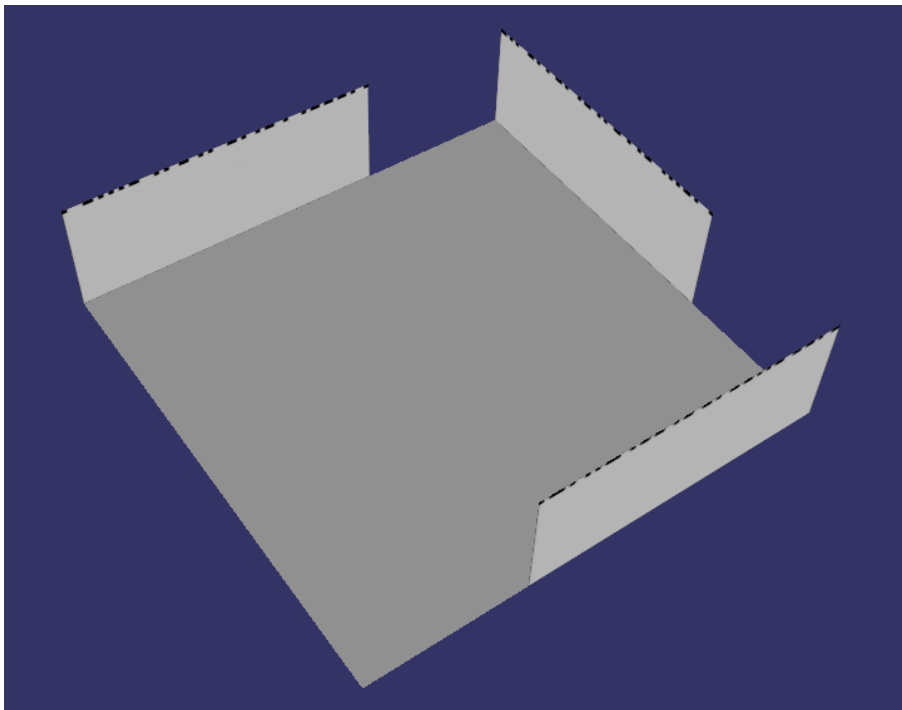


Abbildung 2.2: Ein Modell einer möglichen Bühne

Zum Zeitpunkt dieser Arbeit gab es keine stehende Bühne die man hätte zeigen können. Deshalb wurde nach Vorgabe eine mögliche Installationsanordnung modelliert. In Abbildung 2.2 sieht man von schräg oben eine 12 x 12 m Grundfläche. An drei Seiten stehen so genannte Lichtwände die 3 m hoch und 8 m breit sind.

## 2.3 Lokalisation

Die Lokalisation ist eines der Grundprobleme welches beim Einsatz von mobilen Robotern auftritt. In [Thrun et al., 2006, Seite 193] wird es in drei Teilprobleme zerlegt:

---

<sup>1</sup>Nach der Englischen Bezeichnung: Incremental rotary encoder



**Position Tracking** ist die Verfolgung der Roboterposition und Orientierung bei bekannten Anfangsbedingungen mit Hilfe von Sensordaten. Dabei spielt das Dynamikmodell des Roboters sowie darin modellierte Unsicherheiten eine wichtige Rolle. Bewegt sich der Roboter von einer bekannten Position aus, wird mit dem Dynamikmodell auf Grund der Odometrie seine neue Position geschätzt. Die Unsicherheiten des Modells erzeugen dabei eine Wahrscheinlichkeitsverteilung um diese neue Position, in der sich die wahre Position befinden sollte. Ohne Messungen von weiteren Sensoren die Rückschlüsse auf die Umgebung erlauben, würde die Positionsschätzung mit der Zeit immer ungenauer. Mit Hilfe eines Messmodells lässt sich beurteilen, ob eine Messung an einer bestimmten Position wahrscheinlich erscheint, oder nicht. Dadurch lässt sich die Wahrscheinlichkeitsverteilung der Position nach einer Bewegung durch eine Messung wieder auf den Bereich, in dem der Messwert des Sensors am wahrscheinlichsten ist eingrenzen.

**Global Localization** ist das Finden der Anfangsposition des Roboters unter allen möglichen Posen die im Szenario vorkommen können. Im Vergleich zum *Position Tracking*, bei dem es genügt die Unsicherheit um die geschätzte neue Pose zu berücksichtigen, umfasst hier der Raum möglicher Posen ein erheblich größeres Volumen. Ein Ansatz wäre alle möglichen Posen mit der selben Wahrscheinlichkeit anzunehmen, und mit den ersten Messungen und dem Messmodell diese auf Bereiche einzugrenzen in denen diese Messungen mit hoher Wahrscheinlichkeit auftritt. Das ist viel aufwendiger als Position Tracking und kann bei einem großen Zustandsraum sehr lange Dauern.

**Kidnapped Robot Problem** ist eine verschärfte Variante des *Global Localization* Problems. Dabei geht man davon aus, dass sich der Roboter spontan an einem anderen Ort aufhält als vom Roboter angenommen. Nun müsste die Lokalisation des Roboters wieder im gesamten möglichen Raum erfolgen. Nur das der Roboter diesen Zustand nicht feststellen kann. Um dieses Problem lösen zu können, müsste ein Algorithmus, neben dem Position Tracking, den gesamten Zustandsraum prüfen. Bei einigen Anwendungen würde dies zu lange dauern. So wäre es auch denkbar, dass nicht der gesamte Zustandsraum für jede Messung durchsucht wird, sondern nur Teile davon. Entweder systematisch, so dass nach einer gewissen Zeit ebenfalls der gesamte Zustandsraum durchsucht ist, oder durch zufällige Ziehungen aus dem Gesamtvolumen des Zustandsraums, so dass eine Wahrscheinlichkeit besteht den wahren Zustand zu treffen.

### 2.3.1 Übersicht gängiger Verfahren

Die meisten Lösungsansätze verwenden im Grunde eine Variante des Bayes-Filters wie z.B. in [Thrun et al., 2006, Seite 26] beschrieben. Abhängig vom Einsatzumfeld und der verwendeten Sensoren gibt es aber noch andere Verfahren.

In [Seco et al., 2009] zum Beispiel werden, neben dem Bayes-Filter, drei weitere genannt: *Geometry-Based Methods*, *Minimization of the cost function* und *Fingerprint Methods* die sich in Gebäuden unter Nutzung von Elektromagnetischen Signale verwenden lassen. Je nach Sensorausstattung werden verschiedene Karten der Umgebung verwendet, auf denen sich der Roboter durch Auswertung von Sensormessungen lokalisieren muss. Dabei gibt es statische Karten die vorher angefertigt wurden, die dem Roboter bereits zur Verfügung stehen, und es gibt Karten die der Roboter selbstständig während der Lokalisation erstellen muss. Letzteres wird als *Simultaneous Localization and Mapping* (SLAM) Problem bezeichnet zu dem es bereits viele Veröffentlichungen gibt (u.a.) [Thrun et al., 2006, Seite 309], [Thrun, 2002], [Hertzberg et al., 2012, Seite 229].

Der in Kapitel 1 erwähnte Partikelfilter ist eine Variante des Bayes-Filters. Er ist auch als *Monte Carlo Localization* (MCL) bekannt. Alternativ dazu gibt es zum Beispiel den Kalman Filter, der mit Normalverteilungen die Wahrscheinlichkeiten abbildet. Er ist in seiner Anwendbarkeit auf lineare Probleme beschränkt oder ist auf Linearisierung angewiesen. Er ist allerdings eines der ältesten Verfahren und damit am besten untersucht. In [Ha et al., 2012] wird er bei einem Lokalisationsproblem verwendet. Außerdem gibt es noch so genannte Grid-Based-Filter oder Histogram Filter. Diese teilen den Zustandsraum in so genannte Grids auf in denen ein Wahrscheinlichkeitswert jeweils den Zustand der Region abbildet. Für bestimmte Probleme mit nur zwei Zuständen, kann ein Binary Bayes Filter zum Einsatz kommen. Da in dieser Arbeit ein Partikelfilter verwendet wird, soll seine Funktionsweise erläutert werden.

### 2.3.2 Der Partikelfilter

#### Zustandsraum

Der Partikelfilter bildet die Wahrscheinlichkeitsverteilung des vermuteten Zustandes durch eine diskrete Menge von so genannten Partikeln ab. Ein Partikel ist dabei ein Punkt im Zustandsraum zu dem noch ein Gewichtungswert zugeordnet ist. Ange-

nommen der Zustand wäre die Position in einem dreidimensionalen Koordinatensystem, dann wäre der Zustandsraum alle möglichen Punkte in diesem Koordinatensystem. Ein Partikel in diesem Raum hätte also vier Attribute: einen X-Wert, einen Y-Wert, einen Z-Wert und einen Gewichtungswert. Im Grundzustand, ist der Gewichtungswert bei allen Partikeln gleich und die räumliche Verteilung oder Häufung repräsentiert die Wahrscheinlichkeitsverteilung des vermuteten Zustandes (hier die Position im Koordinatensystem). Häuften sich die Partikel um eine Position, so wäre bei einer zufälligen Ziehung aus den Partikeln, die Wahrscheinlichkeit höher ein Partikel von dieser Position zu ziehen. Mittelt man nun über alle Partikel im Zustandsraum, so erhält man eine Position die dem vermuteten Zustand entspricht. Dabei ist die Varianz über alle Partikel ein Maß dafür, wie zuverlässig diese Position ist.

## Dynamikmodell

Partikelfilter nutzen für die Zustandsschätzung ein Dynamikmodell. In diesem Modell wird abgebildet, wie sich der Zustand über die Zeit verändert, z.B. durch Physikalische Gesetzmäßigkeiten oder durch eine Hilfsgröße die sich messen lässt (z.B. zurückgelegter Weg). Damit erlaubt das Dynamikmodell eine Prognose über den nächsten Zustand abzugeben. Um das Beispiel weiter zu führen stelle man sich vor, man will die Position eines Satelliten der um die Erde kreist bestimmen. Dabei soll lediglich die Position im Raum und nicht seine Orientierung betrachtet werden. Der Zustandsraum besteht damit aus X, Y, und Z-Koordinaten. Für das Dynamikmodell könnten hier die Keplerschen Gesetze zur Bahnberechnung verwendet werden. Allerdings ist es dafür nötig, die Geschwindigkeit des Objektes zu kennen. Dazu wird der Zustandsraum um die Geschwindigkeiten in X, Y und Z-Richtung erweitert. Nun ist es bei einem gegebenen Zustand möglich einen Folgezustand nach einer verstrichenen Zeit zu errechnen. Diese Prognose wird allerdings mit der Zeit immer ungenauer, denn in der Realität gibt es immer Störgrößen, die sich nicht vorhersagen lassen. Häufig jedoch lassen sich Aussagen über die Art und Stärke der Störung machen. Dieses Wissen kann in Form von einer Wahrscheinlichkeitsverteilung um die Prognose verwendet werden. Bei dem Satelliten wäre als Störung der Einfluss von Sonnenwind denkbar der ihn von der Sonne weg beschleunigt. Weiß man wie stark der Einfluss werden kann, könnte man ihn als Statistische Größe in die Prognose einfließen lassen.

## Messmodell

Das Messmodell wird benötigt, um entscheiden zu können, wie plausibel eine Messung bei gegebenem Zustand (Im Beispiel die Position) ist. Ein Sensor, über den das System verfügt, liefert einen Messwert. Mit dem Messmodell kann jetzt zu einem beliebigen Zustand eine Aussage darüber gemacht werden, wie wahrscheinlich es wäre diesen Messwert zu messen. Im Beispiel des Satelliten könnte der Abstand zu einer Bodenstation gemessen werden. Im Messmodell würde dann aus einem gegebenen Zustand, also Position und Geschwindigkeit, zusammen mit weiteren Informationen wie Position der Bodenstation auf der Erde, Datum und Uhrzeit der Messung errechnet werden wie groß der Abstand sein müsste. Je stärker dieser erwartete Abstand von dem gemessenen abweicht, um so unwahrscheinlicher ist die Messung bei diesem Zustand (Position und Geschwindigkeit). Zusätzlich ist natürlich jeder Messwert mit einer Ungenauigkeit behaftet, die vom Messprinzip und Sensortyp abhängt. Aber auch diese lässt sich statistisch angeben und auf den Messwert aufschlagen bevor er ins Messmodell gegeben wird. So lassen sich Zustände im Zustandsraum hervorheben, für die eine gegebene Messung wahrscheinlich sind.

## Angewendet auf den Filter

Wenn der Zustandsraum alle möglichen Zustände eines Systems beschreibt, so sind die Partikel des Partikelfilters eine Untermenge davon, die in ihrer räumlichen Häufung den tatsächlichen Zustand des Systems beschreiben. Während des Betriebs gibt es zwei Ereignisse auf die der Filter reagiert:

Das **Dynamik-Update** berechnet auf Grundlage des Dynamikmodells eine neue Prognose. Konkret wird im Partikelfilter dafür zu jedem Partikel eine eigene Prognose gestellt. Dabei wird eventuelles Systemrauschen durch eine Ziehung pro Partikel aus dessen Wahrscheinlichkeitsverteilung (häufig Gaußverteilung) abgebildet. Die Prognosen aller Partikel bilden dann den neuen Systemzustand.

Der **Mess-Update** Aufruf verarbeitet eine Messung mit Hilfe des Messmodells. Konkret wird für jedes Partikel, und damit den Zustand den es repräsentiert, geprüft wie wahrscheinlich eine solche Messung wäre. Bei einer hohen Wahrscheinlichkeit, wird der Gewichtungswert im Partikel größer. Bei geringer Wahrscheinlichkeit für eine solche Messung sinkt der Gewichtungswert. Nun repräsentieren diese Gewichte der Partikel den neuen Systemzustand.

Ein **Resampling** überführt die Information in den Partikelgewichten wieder in die Partikelhäufung. Es wird normalerweise nach jedem Mess-Update vorgenommen. Dies geschieht durch Ziehen von Partikeln aus der alten Partikelmenge mit einer Wahrscheinlichkeit, die proportional zum Gewicht eines Partikels ist. Somit werden Partikel, die ein hohes Gewicht hatten, häufiger gezogen als jene, die nur ein geringes Gewicht hatten. Damit "überleben" das Resampling die Partikel, deren Zustand von einer vorgenommenen Messung bestätigt wird. Nach der Neuziehung wird das Gesamtgewicht wieder auf 1 normalisiert und alle Partikel erhalten das selbe Gewicht. In [Thrun et al., 2006, Seite 108] wird ein Problem beim Resampling beschrieben, welches bei unabhängigen Ziehungen auftritt. Es wird **VARIANCE REDUCTION** genannt und führt dazu, dass Partikel mit geringem Gewicht zufällig gar nicht gezogen werden. Dies bedeutet einen Informationsverlust, der mit wiederholtem Resampling im Extremfall auf nur einen einzigen Zustand degeneriert. Um dem entgegen zu wirken, sollte das Resampling nur durchgeführt werden, wenn sich die Gewichte in den Partikeln verändert haben. Also nach einer Messung beispielsweise. Weiß man jedoch sicher, dass sich am Zustand nichts ändern kann (z.B. Roboter steht still), sollte man weder Messungen auswerten, noch ein Resampling durchführen. Zudem sollte systematisches Ziehen anstelle von unabhängigen Ziehungen eingesetzt werden. Hierbei wird ein Partikel zufällig ausgewählt, und die übrigen systematisch anhand ihrer Gewichtung ermittelt.

## 2.4 Bildverarbeitung

Da die von einer Kamera aufgenommenen Bilder in dieser Arbeit eine zentrale Rolle spielen, ist es wichtig zu verstehen, wie diese zustande kommen. Daher soll an dieser Stelle kurz darauf eingegangen werden, wie sich eine Kamera formal beschreiben lässt.

Die Lochkamera ist ein sehr einfaches Modell, das häufig zur Veranschaulichung verwendet wird. Es soll hier nicht im Detail beschrieben werden, dafür wären [Jähne, 2005, Seite 203] oder [Tönnies, 2005, Seite 47] zu empfehlen. Doch für das hier vorgestellte Verfahren ist es wichtig, das Konzept dahinter zu verstehen:

Die Kamera projiziert im Wesentlichen Punkte aus dem Raum (3D) auf die Bildebene (2D). Um nun bei gegebenen Koordinaten eines 3D-Punktes in Weltkoordinaten den passenden Punkt auf der Bildebene berechnen zu können, ist die Kenntnis verschiedener Parameter erforderlich. Dabei trennt man in extrinsische und intrinsische Parameter. Unter den extrinsischen Kamera Parametern versteht man die Pose der

Kamera in Weltkoordinaten, also wo sich sie befindet und wie sie ausgerichtet ist. Dies wird in einer Transformationsmatrix ausgedrückt, mit der sich Punkte zwischen dem Kamera- und Weltkoordinatensystem transformieren lassen.

Die intrinsischen Parameter sind nötig, um 3D-Punkte (in Kamerakoordinaten) auf die Bildebene und in Pixelkoordinaten zu projizieren. Sie beinhalten neben der Brennweite, auch die Koordinaten wo die optische Achse auf den Bildsensor trifft. Um die auf den Bildsensor projizierten Punkte auch in Pixeln angeben zu können, gibt es einen Skalierungsfaktor.

Bei realen Kameras, muss in der Regel eine Kalibrierung durchgeführt werden, um diese Parameter zu bestimmen. Dabei können die intrinsischen Parameter gespeichert und wieder verwendet werden, wenn sich die Brennweite nicht ändert (kein optischer Zoom). Die extrinsischen Parameter müssen neu bestimmt werden, sobald sich die Kamera bewegt. Oder die Bewegung muss bekannt sein um sie in die Transformationsmatrix ein fließen zu lassen.

## 3 Lokalisierung mittels Bildverarbeitung

In diesem Kapitel soll das in dieser Arbeit entwickelte Verfahren beschrieben werden, mit dem die Lokalisierung erfolgt. Dabei wird erklärt wie das zu erkennende Bitmuster aufgebaut ist, wie der eingesetzte Partikelfilter ausgelegt wurde, und wie genau der Filter die Bilder beurteilt. Im letzten Abschnitt wird auf verschiedene Parameter eingegangen, die Einstellmöglichkeiten des Filters erlauben.

### 3.1 Bitmuster

Die drei Bitmuster, in der Obersten Zeilen der Lichtwände, sind 64 Bit lang, da eine Lichtwand aus 8 Segmenten mit je 8 Bit aufgebaut ist. Jedes Segment ist 1 m lang. Damit ist ein Bit 125 x 125 mm groß. Als Muster wurde eine Zeichenfolge in Strichcode verwendet. Als Codierung wurde ein Verfahren gewählt, das Code 93<sup>1</sup> genannt wird und von der Firma Intermec entwickelt wurde. Er codiert 48 verschiedene Zeichen in 9 Bit langen Blöcken. Dabei sind mindestens 3 Bit immer **true**(1) und 3 immer **false**(0). Außerdem können höchstens 4 gleiche Bits aufeinander folgen. Code 93 wurde gewählt, weil es schnell zu implementieren war, und sicher stellen konnte, dass es auch in beliebigen Ausschnitten der Muster genug Unterschiede zwischen den Lichtwänden gab. Auf Abbildung 3.1 sieht man die drei verwendeten Codestreifen. Rechts daneben sind die Zeichen für den codierten Abschnitt angegeben. Es sind 9 Bit lange Blöcke. Es gibt 7 solcher

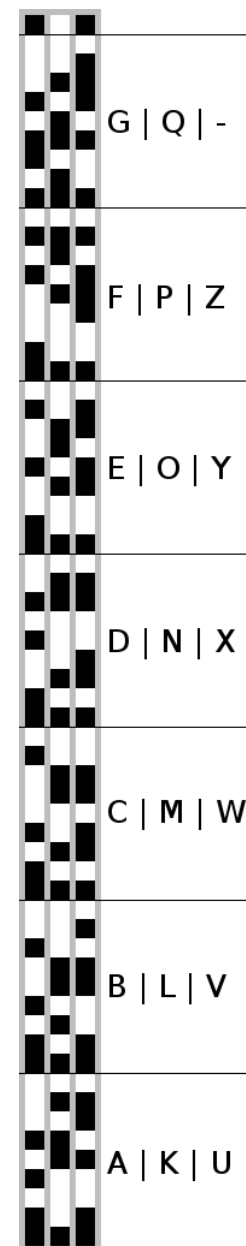


Abbildung 3.1: Muster

<sup>1</sup>Quelle: <http://www.suchymips.de/de/code-93.htm>

Blöcke die 63 Bits füllen, das letzte Bit ist bei zwei schwarz und einem weiß gewählt worden. Eine vollständige Tabelle der Codierung ist im Anhang auf Seite VI zu sehen.

## 3.2 Partikelfilter

Für die Lokalisierung wird ein Partikelfilter verwendet. In diesem Abschnitt wird erläutert, wie dieser entworfen wurde, wie der Zustandsraum gewählt wurde und wie viele Partikel verwendet werden, wie das Dynamikmodell die Odometrie der Antriebe verwendet um die Pose des Roboters zu schätzen und wie im Messmodell die Bilder der Kamera verwendet werden.

### 3.2.1 Zustandsraum

Der Zustandsraum der Partikel setzt sich aus der Position und der Pose des Roboters zusammen. Da er sich ausschließlich auf einer ebenen Bühne befindet, führt dies zur Reduktion der Freiheitsgrade von sechs auf drei:

**x und y** sind die Koordinaten des Roboters auf der Bühne. Der Ursprung ist dabei in der Mitte der Bühne. Die x-Achse zeigt von der Bühnenseite ohne Lichtwand weg und auf die mittlere Lichtwand zu. Die z-Achse zeigt nach Oben und bildet zusammen mit der x- und y-Achse ein Rechtssystem.

**Der Winkel  $\psi$**  repräsentiert die Orientierung des Roboters relativ zur Bühne. Er ist zwischen Roboter x-Achse und Welt x-Achse (Bühne) bei Drehung um die z-Achse.

Bei der Anzahl der Partikel gilt grundsätzlich, dass so viele wie möglich verwendet werden sollten wenn es auf die Genauigkeit des Filters ankommt. Beschränkt wird dies normalerweise von der zur Verfügung stehenden Rechenkapazität und der Zeit in der alle Partikel im Mess-Update verarbeitet werden sollen. Um Echtzeit fähig zu sein, spielt die Zeit eines Mess-Updates die entscheidende Rolle. Unter diesem Aspekt wäre die Anzahl so groß wie nötig, aber so klein wie möglich zu wählen. Für die Simulation sollen 2500 Partikel eingesetzt werden. Dies sind verhältnismäßig wenig, Partikelanzahlen in der Literatur ([Thrun et al., 2006], [Hertzberg et al., 2012]) sind meist im Bereich von mehreren Tausend Partikeln. In der Simulation scheinen aber 2500 Partikel zunächst ausreichend zu sein um die Lokalisation zu ermöglichen. Beim



Einsatz auf einem Roboter wäre sie der Hardware angemessen zu wählen. Hinzu kommt, dass der Algorithmus noch nicht auf Laufzeit optimiert werden konnte und so Simulationsläufe mit hohen Partikelzahlen unnötig lange gedauert hätten.

### 3.2.2 Dynamikmodell

Das Dynamikmodell für den Roboter basiert auf der Odometrie. Als Input bekommt das Dynamik-Update die Encoder-Messwerte des linken und rechten Rades. Es sind absolute Werte aus denen die Differenzen  $\Delta I_{r/l}$  zum letzten Update gebildet werden. Sie werden verwendet, um daraus eine Vorwärtsfahrt  $\Delta s$  und einen Drehwinkel  $\Delta\psi$  zu berechnen:

$$\Delta s = \frac{\Delta I_r + \Delta I_l}{2} \cdot \underbrace{\frac{2\pi r}{g \cdot \gamma}}_E \quad (3.1)$$

$$\Delta\psi = \frac{\Delta I_r - \Delta I_l}{2} \cdot \frac{2 \cdot E}{D} \quad (3.2)$$

mit Radabstand  $D$ , Getriebeübersetzung  $g$ , Radradius  $r$  und Geberauflösung  $\gamma$

Jedes Partikel berechnet daraus seinen neuen Zustand:

$$x_t = x_{t-1} + \cos(\psi_{t-1} + \frac{\Delta\psi}{2}) \cdot \Delta s \quad (3.3)$$

$$y_t = y_{t-1} + \sin(\psi_{t-1} + \frac{\Delta\psi}{2}) \cdot \Delta s \quad (3.4)$$

$$\psi_t = \psi_{t-1} + \Delta\psi \quad (3.5)$$

Dabei wird rechnerisch erst eine Drehung um  $\frac{\Delta\psi}{2}$  vollzogen, gefolgt von der Geradeausfahrt um  $\Delta s$  mit einer abschließenden Drehung um  $\frac{\Delta\psi}{2}$ . Damit der Partikelfilter funktioniert, muss er die Messunsicherheiten der Eingangswerte berücksichtigen. Dazu wird vor der Zustandsberechnung, zu  $\Delta s$  und  $\Delta\psi$  ein Gaußsches Rauschen addiert. Es ist proportional zu deren Betrag:

$$\Delta s_{err} = \Delta s \cdot \sigma_s \cdot \text{RandomGaussian}() \quad (3.6)$$

$$\Delta\psi_{err} = \Delta\psi \cdot \sigma_\psi \cdot \text{RandomGaussian}() \quad (3.7)$$

$\sigma_s$  und  $\sigma_\psi$  sind dabei ein Maß dafür wie breit die Streuung der Normalverteilung ist. Sie sind Parameter die auf den Anwendungsfall, nach Stärke des erwarteten Rauschens, eingestellt werden müssen. Dabei soll die Streuung der Partikel im Zustandsraum mindestens genau so groß sein, wie die Streuung um den wahren Wert, verursacht durch Messunsicherheit der Sensoren. Wenn das  $\sigma$  zu klein gewählt, so

kann es passieren, dass die Verteilung der Partikel den wahren Zustand nicht mehr enthält. Somit gibt es bei einer Messung kein Partikel mehr, dessen Zustand diese als wahrscheinlich erscheinen lässt. Damit folgen die Partikel im Zustandsraum einer falschen Schätzung, und die Messungen sind wertlos. Der Partikelfilter hätte die Position verloren.

Setzt man das  $\sigma$  größer an, so divergieren die Partikel mit jedem Dynamik-Update stärker und der wahre Wert wird hoher Wahrscheinlichkeit von Partikeln abgedeckt, so dass bei einer Messung diese einen guten Score bekommen und durch ein Resampling sich die Partikel wieder um den wahren Wert konzentrieren. Allerdings ist bei zu großem  $\sigma$  die Aussagekraft der Partikelverteilung sehr ungenau und es sind viele Partikel nötig, um die nötige Dichte im Zustandsraum zu gewährleisten. Dabei spielt es eine entscheidende Rolle, wie häufig Messungen erfolgen. Denn zwischen den Messungen muss sich der Filter auf das Dynamikmodell verlassen, und bei großem  $\sigma$  ist die Schätzung nach wenigen Schritten bereits mit einer großen Unsicherheit verbunden.

### 3.2.3 Messmodell

Als Messungen werden die Bilder einer Kamera auf dem Roboter verwendet. Das Messmodell dahinter beruht auf dem Wissen um die Position der erwähnten Mustern in der Umgebung. Dies kann als Karte der Umgebung verstanden werden, anhand derer sich der Roboter orientieren muss. Es gibt drei verschiedene Muster, auf jeder Lichtwand eines. Die Muster werden in der obersten Zeile der Lichtwand angezeigt, um möglichst selten verdeckt zu werden. Bei dem Messmodell gilt es nun zu prüfen, ob ein Bild zu einer bestimmten Pose passt oder nicht. Dafür könnte man in dem Bild nach den bekannten Mustern suchen, und sobald diese gefunden sind versuchen diese einer Pose zuzuordnen. Aber eine solche Mustersuche in einem Bild ist immer in verschiedene Schritte aufgeteilt, die aufeinander aufbauen. Also z.B. Binarisierung über einen Schwellwert, Regionenbildung mit Charakterisierung und anschließende Auswertung ausgewählter Regionen. Oder Kantenerkennung, Hough-Transformation und finden von parallelen kurzen Linien. Ein Problem daran ist, dass wenn in einem ersten Schritt z.B. ein Schwellwert falsch gewählt wurde, oder nur sehr schwache Kanten vorhanden sind, alle folgenden Schritte scheitern, weil ihre Vorbedingungen nicht ausreichend erfüllt werden. Aus diesem Grund wurde ein anderer Ansatz verfolgt, bei dem man nicht das Bild und die Informationen darin als Ausgangspunkt nimmt, sondern die Pose der Partikel und die Position der Muster im Raum. Dazu wird aus den Koordinaten des Roboters auf der Bühne und

dessen Ausrichtung die Pose der Kamera berechnet. Und anschließend die Position des Musters aus dem Raum in Pixelkoordinaten projiziert werden. Damit könnte man für jede beliebige Pose des Roboters sagen wo im Bild das Muster zu sehen sein müsste und diese Bereiche mit dem erwarteten Muster vergleichen. Je besser der Bereich zu dem Muster passt, umso höher wird der Score für die Partikelbewertung. Wie die gefundenen Pixel im Bild mit dem erwarteten Muster verglichen werden, wird in Abschnitt 3.3 näher beschrieben. Um die Position des Musters im Bild aus der Roboter Position zu berechnen sind mehrere Koordinatentransformationen und eine Projektion nötig. Die Musterposition liegt als Punktmenge  $M_W$  der Pixelmittelpunkte in Weltkoordinaten vor. Um sie mit der Kameragleichung in das Bild zu projizieren, müssen sie in die Kamerakoordinaten transformiert werden. Dazu sind folgende Schritte nötig:

**Welt zu Roboter** ( $T_R^W$ ) In diese Transformation fließt die Pose des Roboters ein, die in Weltkoordinate vorliegt. Diese Transformation besteht aus Translation in x- und y-Richtung sowie einer Drehung um die z-Achse mit dem Winkel  $\psi$ . Sie muss für jedes Partikel neu erzeugt werden. Da die Pose sich natürlich ständig ändert, und unter den Partikeln verschieden ist.

$$T_R^W = \begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 & 0 \\ \sin(\psi) & \cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3.8)$$

**Roboter zu Kamera** ( $T_K^R$ ) Diese Transformation entspricht der extrinsischen Kameramatrix, die die Pose der Kamera relativ zum Roboter ausdrückt. In der Simulation wurde sie in 500 mm Höhe am Roboter angebracht. Sie blickt in Fahrtrichtung und ist  $30^\circ$  nach Oben geneigt. Ist diese Transformation einmal bekannt, so kann sie immer wieder verwendet werden. In dieser Arbeit ist sie aus der Simulation bekannt. Aber bei einer realen Anwendung müsste zunächst eine Kamerakalibrierung durchgeführt werden, um sie zu bestimmen. Um dieses Verfahren erst einmal untersuchen zu können, wird auf die Problematik der Kamerakalibrierung in dieser Arbeit nicht weiter eingegangen.

Durch Multiplikation der Transformationsmatrizen

$$M_K = T_K^R \cdot T_R^W \cdot M_W \quad (3.9)$$

erhält man die Koordinaten der Bitmuster im Kamerasystem  $M_K$ . Diese können nun mit einer perspektivischen Projektion

$$P \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{pmatrix} \quad (3.10)$$

und der intrinsischen Kameramatrix

$$K_i = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (3.11)$$

in eine Punktmenge in Pixelkoordinaten  $M_P$  projiziert werden

$$M_P = K_i \cdot M_K \quad (3.12)$$

### 3.2.4 Initialisierung

Beim Start der Lokalisation muss den Partikeln eine Anfangsposition zugewiesen werden. Dies ist das in Abschnitt 2.3 beschriebene Teilproblem der Globalen Lokalisation. Eine übliche Vorgehensweise um einen sinnvollen Anfangszustand zu erhalten, ist die erste Messung zu verwenden und den gesamten Zustandsraum nach korrespondierenden Zuständen zu durchsuchen. Da ein durchsuchen des gesamten Raumes sehr lange dauern würde, wurde dieser eingegrenzt und damit eine Forderung an die Anfangs Pose des Roboters gestellt. Für die anfängliche Globale Lokalisierung muss der Roboter das Bit-Muster der mittleren Lichtwand sehen können. Es wird gefordert, dass er in Richtung x-Achse ausgerichtet ist mit einer Toleranz von  $\mp 20^\circ$  der Blickrichtung. An die genaue Position auf der Bühne wird keine weitere Anforderung gestellt, solange das Bit-Muster im Bild ist, und der Blickwinkel in der Toleranz ist. In einer realen Anwendung würde der Roboter etwa mittig auf die Bühne gestellt und in Richtung der mittleren Lichtwand gedreht werden. Diese Globale Lokalisation würde ein paar Minuten brauchen und danach könnte der Roboter seinen programmierten Weg abfahren. Da die Lokalisation nur einmal zu Beginn stattfinden muss, sollte dieses Verfahren kein Problem für die von BBM beschriebene Anwendung sein.

### 3.2.5 Schätzung aus Partikeln berechnen

Der Algorithmus soll die Position des Roboters auf der Bühne, sowie dessen Orientierung angeben können. Weil der Partikel-Filter diese Information nur in der Partikelverteilung im Zustandsraum kennt, muss daraus eine geschätzte mittlere Position berechnet werden. Um einschätzen zu können, wie verlässlich diese Positionsangabe ist, wird außerdem noch die Varianz berechnet. Für die Position, als zweidimensionale Größe, wird sie als Ellipse um die Position ausgegeben und besteht aus großer Halbachse, kleiner Halbachse und Winkel zur x-Achse des Bühnenkoordinatensystems. Die Ellipse drückt den Vertrauensbereich aus, der  $3\sigma$  einer Normalverteilung entspricht. Für den Winkel wird neben dem Mittelwert die Standardabweichung berechnet. Damit steht die Mittlere Pose des Roboters über alle Partikel samt Standardabweichung zur Verfügung.

## 3.3 Musterbewertung

Bei der Musterbewertung geht es darum, zu einer bekannten Region im Bild eine Bewertung ab zu geben, wie gut sie zu einem gegebenen Muster passt. Bei der Bewertung wird zu jedem Bit des Musters geprüft, ob an dessen Position ein schwarzer oder ein weißer Wert vorliegt. Dabei soll ein Score berechnet werden der zwischen 0 und 1 liegt. Wobei 0 keinerlei und 1 volle Übereinstimmung signalisiert. Es ist gewollt, dass hier keine harte Entscheidung getroffen wird. Denn damit gehen Informationen verloren, die zur Lokalisierung beigetragen hätten. Wenn beispielsweise ein Teil des Musters verdeckt wurde, so wäre immer noch ein mittlerer Score möglich. Bei einer harten Entscheidung würde Partikel zu denen das Bild normalerweise passt genau so schlecht bewertet wie alle anderen.

### 3.3.1 Spezialfall: Projektion ergibt keine Bildpunkte

Für die Musterbewertung stehen die Bildpositionen jedes Bits der drei Bitmuster aus der Projektion zur Verfügung. Dabei entspricht sie stets dem Mittelpunkt der  $125 \times 125$  mm großen Bits der Lichtwände. Der erste Schritt ist eine einfache Überprüfung, ob die Pixelkoordinaten der Muster-Bits tatsächlich in dem Bild liegen. Denn die Projektion ist nicht auf einen Bildausschnitt beschränkt. Aus der Menge der Bildpunkte die ein Muster-Bit repräsentieren  $M_P$  können so alle aussortiert werden, die nicht in dem betrachteten Bildausschnitt gelandet sind. Ein Spezialfall

wäre nun, dass kein Muster-Bit als Punkt im Bild gelandet ist. In diesem Fall wird ein fester Score gegeben, der nicht 0 sein darf. Genau 0 wäre für den Partikelfilter schlecht, da dieser das Gewicht des Partikels mit dem Score multipliziert um es zu bewerten. Es gibt hier zwei Fälle die zu unterscheiden sind:

**Wahrer Zustand lässt kein Muster zu:** Der Roboter und damit die Kamera stehen so zu den Lichtwänden, dass sich kein Bit-Muster im Bild befindet. Damit gibt es, bei Partikeln die dieser Ausrichtung entsprechen, keine Punkte im Bild die sich auswerten lassen. Würde man einen Score von 0 geben, so würde der richtige Zustand, wegen der Multiplikation und anschließendem Resampling, gelöscht werden! Dies ist nicht erwünscht. Eine mögliche Lösung wäre in diesem Fall das Partikel mit einem mittlerem Score zu bewerten um den richtigen Zustand zu schützen. Ein zu hoher Score wäre jedoch im Folgenden Fall ungünstig.

**Falscher Zustand lässt kein Muster zu:** Ein Partikel, dessen Zustand stark von der wahren Pose abweicht, führt zu keinen Bildpunkten für das Bit-Muster. Im Bild befindet sich aber ein Bit-Muster. Deshalb müsste es andere Partikel geben, deren Zustände zum Bild passen und nicht automatisch einen festen Score bekommen. Würde nun, wie im ersten Fall vorgeschlagen, ein mittlerer Score gegeben, würde dieses Partikel überbewertet werden. Im Vergleich zu Partikeln dessen Zustand ein Bit-Muster an falscher Stelle vorhersagt, wäre dieser Fall erheblich besser bewertet. Auch wenn ein kleiner Wert gewählt wird, könnte eine ungünstige Vorhersage bei einem anderen Partikel einen schlechteren Score ergeben. Angestrebt wäre aber, dass dieser zweite Fall möglichst schlecht bewertet wird.

Deshalb wird ein Minimalscore von 0,01 festgelegt der in diesen Fällen gegeben wird. Zudem wird auch bei vorhandenem Bildpunkten für Bit-Muster bei dessen Auswertung mindestens dieser Minimalscore gegeben. Dadurch werden beim ersten Fall, bei dem kein richtiges Bit-Muster zum Auswerten im Bild ist, alle Partikel gleich bewertet. Da es kein Partikel gibt, das einen hohen Score bekommen kann. Es muss nur sichergestellt werden, dass die Auswertung beliebiger Bildpunkte als Bit-Muster den Minimalscore möglichst nicht übertrifft. Um dies zu prüfen, wurde zu einem Testbild (siehe Bild 3.2) für 1,8 Millionen Posen, die gleichmäßig über den Zustandsraum verteilt waren, der Score berechnet. Nur 136 davon lagen über dem Minimalscore von 0,01. Wie in Bild 3.2 unten zu sehen, gibt es keine starke Häufung der Scores die über dem Minimum liegen, zudem ist kein Score über 0,25. Eine so geringe Anzahl von Überschreitungen des Minimalscores wird als akzeptabel

angesehen.

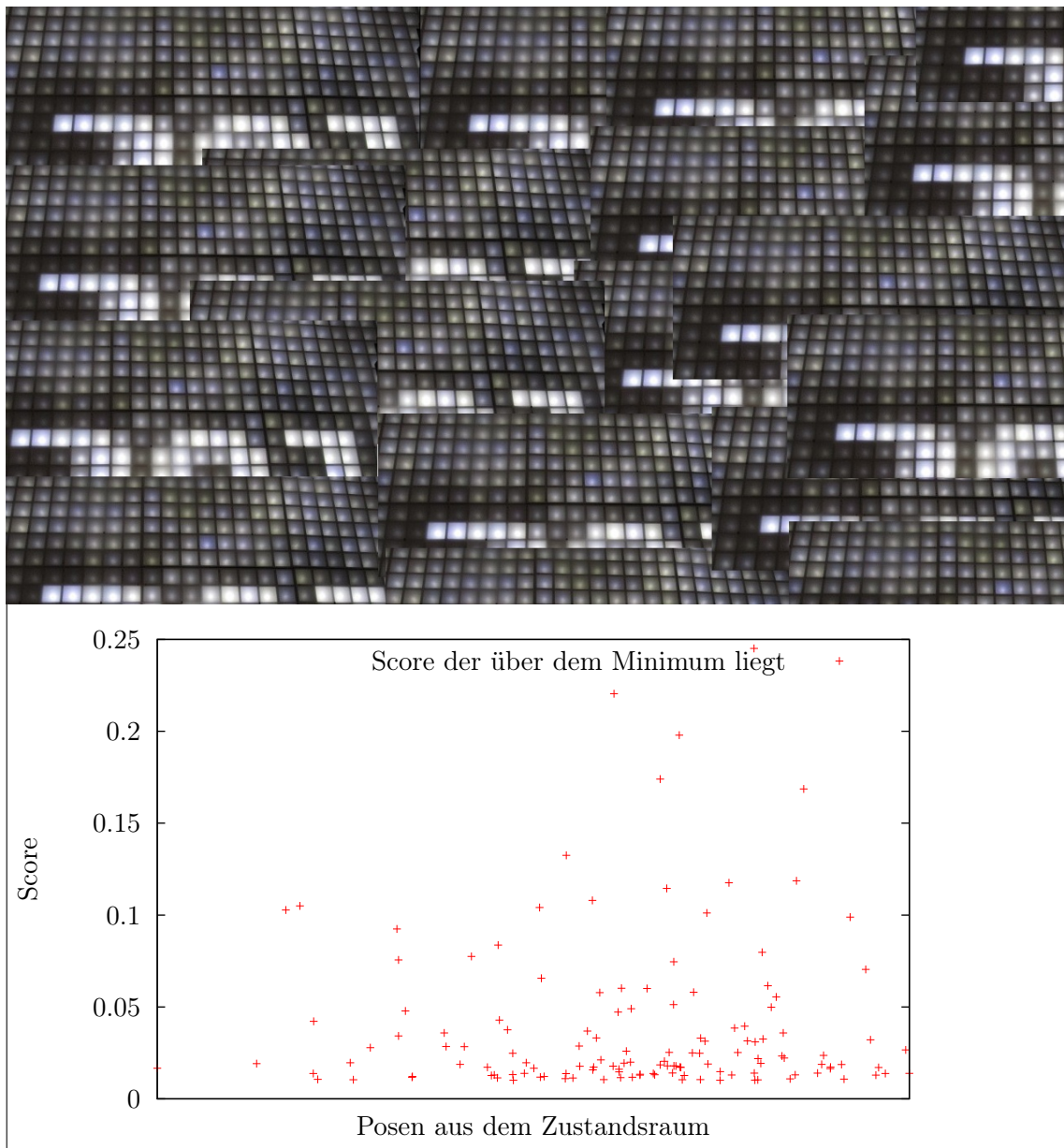


Abbildung 3.2: 1,8 Millionen Posen wurden mit dem Bild oben Bewertet. Nur 136 davon liegen über dem Minimalscore. Sie sind hier abgebildet.

Werden alle mit dem selben Score bewertet, bleibt das Gewicht der Partikel ebenfalls gleich. Dies führt beim Resampling dazu, dass die Partikelverteilung erhalten bleibt. Beim zweiten Fall sollte es andere Partikel geben, deren Zustand zu dem Bild passt und die damit einen höheren Score bekommen. Wäre dies nicht der Fall, so wäre die Lokalisation fehlgeschlagen. Dadurch, dass der Minimalscore die geringste Bewertung ist die ein Partikel bekommen kann, gibt es das Problem einer Überbewertung nicht mehr.

### 3.3.2 Mindestanzahl Auswertbarer Bildpunkte

Auch wenn es Punkte im Bild gibt, die mit dem Bit-Muster ausgewertet werden können, so erfolgt dies erst ab einer gewissen Anzahl. Für die Mindestanzahl auswertbarer Punkte wurde 18 gewählt. Damit soll verhindert werden, dass zufällig wenige Punkte auf Bildbereichen liegen, die mehr oder weniger gut zum gesuchten Bit-Muster passen. Außerdem ist mit 18 Punkten sichergestellt, dass keine Teile des Bit-Musters von einem Streifen auf einem anderen gefunden werden. Da die Codebausteine einzeln immer 9 Bits lang sind und im Gesamtmuster einzigartig. Wird diese Mindestanzahl nicht erfüllt, ergibt sich ebenfalls der Minimalscore.

### 3.3.3 Bestimmung mittlerer Weiß- und Schwarzwerte

Um einen Vergleich anstellen zu können, ob ein Bit im Bild schwarz oder weiß ist, sollen keine absoluten Weiß- und Schwarzwerte verwendet werden. Stattdessen wird über alle weißen Bits im Bild ein mittlerer Weißwert  $\hat{h}_w$  errechnet:

$$\hat{h}_w = \frac{1}{n} \sum_{i=1}^n h_w(i) \quad (3.13)$$

Für den Schwarzwert  $\hat{h}_s$  wird genauso verfahren. Aus diesen beiden Größen wird außerdem die Differenz  $\kappa$  gebildet:

$$\kappa = \hat{h}_w - \hat{h}_s \quad (3.14)$$

An ihr lässt sich prüfen, ob sie negativ oder Null ist. Es würde darauf schließen lassen, dass ein falsches (oder kein) Muster vorliegt. Trifft dies zu, so wird der Minimalscore vergeben.

### 3.3.4 Bewertungskriterium

Für den Score  $P$  der übrigen Bit-Musterpunkte wird ein quadratischer Abstand in der Helligkeit  $\Delta h_i$  vom jeweiligen Referenzwert  $\hat{h}$  verwendet:

$$\Delta h_i = (h_i - \hat{h}_{w/s})^2 \quad (3.15)$$

abhängig davon, ob im Muster ein schwarzes oder ein weißes Bit erwartet wird. Je größer dieser Abstand wird, umso schlechter passt dieses Bit aus dem Bild in das Bit-Muster. Würde man diesen Abstand direkt in den Score einfließen lassen, gäbe es bei sehr kontrastarmen Bildern ein Problem. Da der Vergleichswert für



Schwarz und Weiß aus dem Bild berechnet wird, wären sie bei einfarbigen Flächen identisch. Denkbar wäre auch, dass eine solche einfarbige Fläche einen relativ hohen Score bekommt. Dies wäre bei ungünstigem Bildrauschen möglich, bei dem zufällig schwarze Bits dunkler und weiße Bits heller sind. Durch den geringen Kontrast sind die Abstände nur sehr klein, dies würde zu einem hohen Score führen. Da dieser Abstand bei sinkendem Kontrast immer schwächer wird, fließt zusätzlich noch  $\kappa$  mit in die Berechnung für den Score eines Bits  $p_i$  ein:

$$p_i = \Delta h_i \cdot \frac{\kappa_{min}^2}{\kappa^2} \quad (3.16)$$

mit  $\kappa_{min}$  als Parameter lässt sich einstellen bis zu welcher Größe sich  $\kappa$  negativ auf den Score auswirkt. Und ab wann es den Score positiv beeinflusst. Ein Wert von 20 für  $\kappa_{min}$  reicht aus, um den Gesamtscore des Musters bei geringem Kontrast deutlich zu senken und Hilft bei gutem Kontrast den Score zu verbessern. Nach diesem Verfahren wird für jedes Bit ein solcher Score berechnet und der Mittelwert daraus gebildet:

$$\hat{p} = \frac{1}{n} \sum_{i=1}^n (h_i - \hat{h}_{w/s})^2 \cdot \frac{\kappa_{min}^2}{\kappa^2} \quad (3.17)$$

Die Bildung des Mittelwertes ist nötig, damit der Score unabhängig von der Anzahl der sichtbaren Bits bleibt. Eine einfache Summe würde dazu führen, dass viele Bits einen schlechteren Score bekommen als wenige. Der letzte Schritte der Bewertung ist eine Exponentialfunktion mit negativem Exponenten:

$$P = e^{-\frac{\hat{p}}{2}} \quad (3.18)$$

Sie führt bei sehr kleinem  $\hat{p}$  zu einem Score nahe 1, während bei großem  $\hat{p}$  der Score gegen 0 geht. Dieses Verhalten ist nötig um den Score mit dem Gewicht der Partikel multiplizieren zu können. Sie stellt sicher, dass der Score  $P$  immer im Intervall  $[0..1]$  bleibt

### 3.4 Parameter einstellen

An dieser Stelle soll beschrieben werden, mit welchen Größen das Verhalten von Filter und Bildbewertung beeinflusst werden können. Für den Filter lassen sich folgende Variablen verändern:

**Anzahl der Partikel** ( $N_p = 2500$ ) Wie in Abschnitt 3.2.1 schon beschrieben werden den 2500 Partikel eingesetzt. Je nach Anwendung muss ihre Anzahl angepasst werden. Sie ist dabei besonders von den Rechenkapazitäten abhängig die zur Verfügung stehen, sowie den zeitlichen Anforderungen.

**Raddurchmesser** ( $R = 80mm$ ) Der Filter nutzt diesen Parameter um aus Radumdrehungen eine zurückgelegte Strecke zu berechnen. Dies ist für die Berechnungen im Dynamik-Update relevant.

**Getriebeübersetzung** ( $g = 14,5$ ) Die Getriebeübersetzung wird vom Filter verwendet, um von Umdrehungen des Encoders in Radumdrehungen um zu rechnen. Dabei ist ein Verhältnis  $g : 1$  bzw. Encoder:Rad gemeint.

**Auflösung Drehimpulsgeber** ( $\gamma = 2000 \frac{1}{U}$ )

**Impulse pro Meter** ( $I_m = 57693.67$ ) Diese Größe kombiniert mehrere Maße und Parameter des Antriebs und des Encoders. Mit ihr lässt sich aus einer Anzahl von Impulsen des Encoders die gefahrene Strecke berechnen. Darin enthalten sind:

$$I_m = \frac{g \cdot \gamma}{2\pi R} = \frac{14,5 \cdot 2000}{2\pi \cdot 0,08m} = 57693,67 \frac{1}{m} \quad (3.19)$$

**Abstand zwischen den Rädern** ( $D_R = 700mm$ ) Dieser Parameter wird verwendet, um bei Drehungen den zurück gelegten Weg der Räder berechnen zu können. Im Filter wird aus Differenzen des linken und rechten Encoders damit ein Drehwinkel bestimmt.

**Unsicherheits Koeffizient Strecke** ( $\sigma_s = 0.2$ ) Diese Größe steuert wie groß das Rauschen ist welches auf die Bewegung eines Partikels wirkt. Dabei ist  $\sigma_s$  eine Prozentgröße. Sie wird mit dem zu verrauschenden Wert und der Ziehung einer Gaußschen Normalverteilung multipliziert.

**Unsicherheits Koeffizient Drehung** ( $\sigma_\psi = 0.2$ ) Dieser Koeffizient ist ebenfalls eine prozentuale Größe wie  $\sigma_s$ . Er bezieht sich jedoch nur auf Drehungen. Er wird genau so verrechnet wie der Koeffizient für eine Strecke.

PicRating:

**Mindestanzahl an sichtbaren Bits** ( $B_{min} = 18$ ) Bei der Bildauswertung werden Partikel automatisch mit dem Mindestscore bewertet, wenn weniger als  $B_{min}$  Positionen für Bit-Muster berechnet wurden. Dies dient vor allem dem Schutz vor fehlerhaften hohen Scores durch zufälliges Übereinstimmen weniger Pixel im Bild.

**Kontrast Grenzwert** ( $K_T = 20$ ) Wenn der mittlere Kontrast über alle Pixel in den Bit-Musterpositionen kleiner wird als  $K_T$ , wird der Score abgewertet. Ist er

größer, wird er verbessert.

**Größe Pixelgrid** ( $U_{Pix} = 1$ ) An den Positionen wo ein Bit-Muster vermutet wird, können unterschiedlich viele Pixel ausgewertet werden. Mit  $U_{Pixel} = 0$  würde nur ein Pixel an der Position betrachtet werden. Der wert gibt an wie viele Nachbar Pixel in jede Richtung mit in die Berechnungen einbezogen werden sollen. Ein Wert von 3 würde ein 7 x 7 Raster mit 49 Pixeln bedeuten.



## 4 Die Simulationssoftware

Die Software die im Rahmen dieser Arbeit entwickelt wurde, ist in C++ geschrieben. Als Entwicklungsumgebung wurde Eclipse mit den *C/C++ Development Tools* (CDT) auf einem Linux<sup>1</sup> Betriebssystem verwendet.

Grundlage der Simulation ist das 3D Grafiktoolkit *Open Scene Graph* (OSG)<sup>2</sup>. Damit lässt sich eine 3D Szene in Form eines Graphen aufbauen und mit einem Viewer darstellen. Um den Ablauf kontrollieren zu können, lässt sich die Render-Schleife manuell aufrufen um jeden Frame einzeln berechnen zu lassen. Dies wurde als Simulationsschritt gewählt in dem alle nötigen Berechnungen durchgeführt werden können. Da die Geschwindigkeit mit der die Simulation im manuellen Modus abläuft nicht begrenzt wird, wurde eine Mindestbearbeitungszeit integriert. Denn die Geschwindigkeiten von Objekten in der Simulation, wie dem Roboter, werden durch eine zurückgelegte Strecke pro Simulationsschritt festgelegt. Bei sehr schneller Hardware ergab dies eine zu hohe Bewegungsrate um den Roboter noch manuell steuern zu können. Für automatisierte Simulationsläufe mit festgelegten Fahrprofilen könnte man diese Begrenzung wieder lösen um Zeit zu sparen.

### 4.1 Koordinatensysteme

OpenGL und OpenCV definieren die Koordinatensysteme für eine Kamera sehr unterschiedlich. Um die Pose der Kamera aus der Simulation (OpenGL Koordinatensystem) in die extrinsischen Kamera Parameter zu transformieren, wie OpenCV sie versteht, muss bekannt sein wie diese beiden Systeme zum Weltkoordinatensystem in der Simulation orientiert sind. In Abbildung 4.1 sind alle drei Koordinatensysteme eingezeichnet. Kamera OpenCV Kamera OpenGL Simulation

---

<sup>1</sup>Ubuntu 12.04 LTS

<sup>2</sup><http://www.openscenegraph.org/>

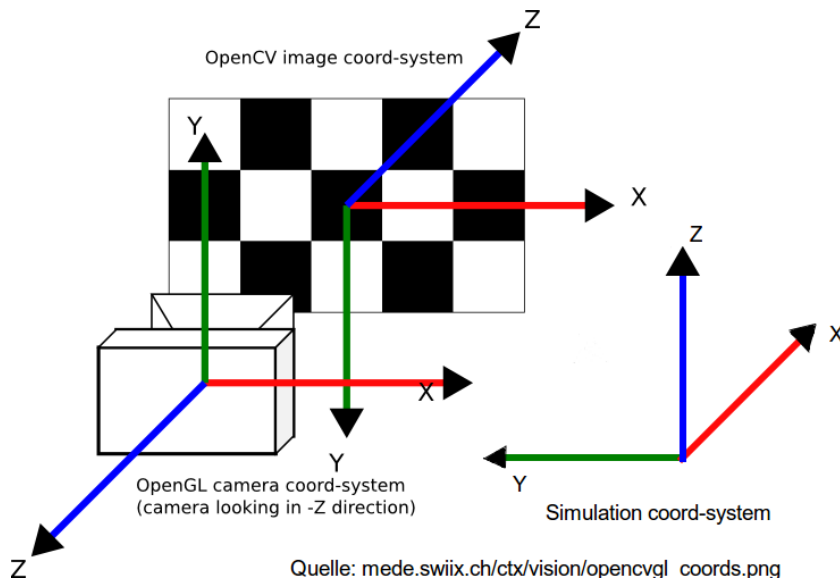


Abbildung 4.1: Koordinatensysteme in der Simulation

## 4.2 Die Szene

Die Szene in der Simulation ist aus mehreren Komponenten aufgebaut, die im Folgenden näher beschrieben werden. Dabei wird ein Vergleich zu den echten Elementen auf der Bühne gezogen, um zu erläutern wie deren Attribute in der Simulation abgebildet werden können. In Abbildung 4.2 kann man alle Elemente der Szene erkennen.

**Die Grundfläche (1)** der Bühne misst 12 x 12 m. Sie wird als einfach weiße Fläche in der Szene dargestellt. Da die Bildverarbeitung nur auf den Oberen Teil des Bildes beschränkt ist, spielen Farbe und Helligkeit keine Rolle bei der Erkennung des Musters im Bild.

**Die Lichtwände (2,3)** sind zu drei Seiten der Grundfläche aufgestellt. Sie messen 3m in der Höhe und 8m in der Länge. Wie bereits in Abschnitt 3.1 beschrieben soll das Muster in der obersten Zeile der Lichtwände dargestellt werden. Hierzu können verschiedene Texturen geladen werden die das Bitmuster enthalten. In der Realität sind die Lichtwände auf der Bühne aus 1 x 1 Meter großen Segmenten zusammen gesetzt, dies wird in der Simulation nicht abgebildet. Es wäre jedoch denkbar, dies in den Aufbau der Texturen einfließen zu lassen durch leichte Abstandsänderungen der Bits in der Textur. Auf jedem Segment sind 8 x 8 Bits unter gebracht. Das macht eine Kantenlänge von 125 mm bei jedem dieser Pixel und 64 Pixel über die ganze Länge einer Wand. Ein solches Segment besteht aus einer Milchigen Plexiglasplatte, die auf eine

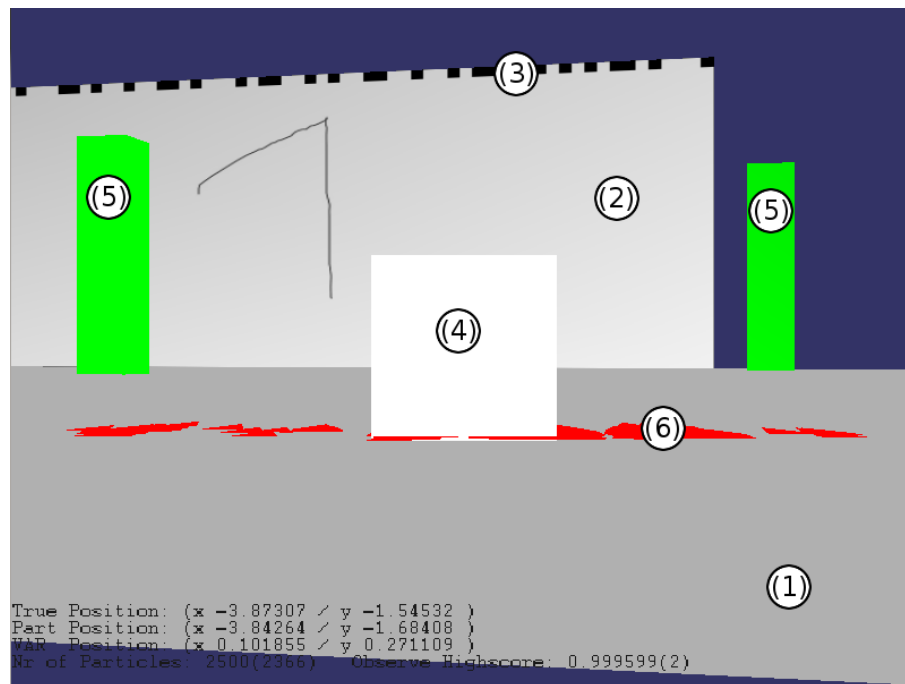


Abbildung 4.2: Simulierte Szene

Struktur geschraubt wurde die für jedes Pixel ein Leuchtmittel vorsieht. Dessen Helligkeit lässt sich einstellen, wäre aber für das Bitmuster auf die Zustände: dunkel(ausgeschaltet) oder hell(ein, mit größter Helligkeit) einzustellen. Die Plexiglasplatte wird also pro Pixel von hinten durchleuchtet. Dies führt dazu, dass die Helligkeitsverteilung in einem beleuchteten Pixel inhomogen ist. In der Mitte ist die größte Helligkeit, während sie radial nach außen etwas abnimmt. Die Ecken der Pixel sind die dunkelsten Stellen. Auf Abbildung 1.2 auf Seite 2 ist dieser Effekt gut zu sehen. Er wird besonders stark, wenn die Leuchtmittel mit niedriger Helligkeit betrieben werden. Bei hoher Helligkeit ist der Effekt noch wahrnehmbar, aber nicht mehr so ausgeprägt. Und noch etwas fällt auf wenn man dieses Bild betrachtet. Die Lichtfarbe und Helligkeit variiert leicht von Pixel zu Pixel. All diese Effekte können in der Simulation durch verändern der Texturen abgebildet werden.

**Der Roboter (4)** wird nur durch einen weißen Würfel dargestellt. Die genaue Form und das Aussehen spielen für die Lokalisation keine Rolle. Die Kameras sind so angebracht, dass der Roboter selbst nicht im Bild zu sehen ist. Der Würfel ist für den Bediener gedacht, der die Szene aus einem Blickwinkel hinter dem Roboter steuert. Das Bild 4.2 ist aus diesem Blickwinkel aufgenommen. Der Roboter lässt sich mit den Tasten W, A, S, D (vorwärts, links, rückwärts, rechts) grob verfahren oder mit einem Pad<sup>3</sup> mit Analogsticks auch präziser steuern.

---

<sup>3</sup>PlayStation 3 Wireless Sixaxis Controller

Die Steuerbefehle für den Roboter werden als Geschwindigkeit (gerade aus) und Drehrate interpretiert. Diese führen dann, in jedem Simulationsschritt zu einer Positionsänderung. Es wird erst eine Drehung um die Hälfte der Winkeländerung durchgeführt. Anschließend wird in die neuer Richtung geradeaus die Streckenänderung zurück gelegt. Und mit einer zweiten Drehung bis zur vollen Winkeländerung wird die Bewegung abgeschlossen.

**Besucher (5)** werden mit 400 x 400 x 1800 mm Quadern dargestellt. Damit sie für den Bediener besser in der Szene zu erkennen sind, wurden sie grün gefärbt. Sie bewegen sich nicht, sondern stehen an vorher festgelegten Positionen. Vor einer Simulation wird die Anzahl der Besucher eingestellt, die auf der Bühne stehen sollen. Dabei gibt es 25 mögliche Positionen aus denen dann die gewünschte Anzahl zufällig gezogen wird. Sie sollen mögliche Verdeckungen im Bildbereich des Roboters darstellen. Damit soll untersucht werden, wie gut der Algorithmus mit teilweise verdeckten Code-Bereichen zurecht kommt.

**Partikel (6)** werden zur Veranschaulichung und zu debugging Zwecken visualisiert. Sie werden durch rote kleine spitze Dreiecke dargestellt. Der spitze Winkel zeigt dabei die Orientierung an. Sie befinden sich nur auf dem Boden und beeinträchtigen die Bildverarbeitung deshalb nicht.

## 4.3 Messen in der Simulation

Die Simulation soll, neben der Visualisierung, Messungen liefern um den Lokalisationsalgorithmus testen zu können. Anders als bei Messungen an realen Experimenten hat man in der Simulation den Vorteil, alle das Messergebnis beeinflussenden Faktoren unter Kontrolle zu haben. Möchte man also, dass Messungen eine systematische Abweichung aufweisen, muss diese in der Simulation definiert werden. Genau so verhält es sich mit statistischen Abweichungen. Es lassen sich also schnell die Messunsicherheiten der simulierten Vorgänge anpassen. Beim Debugging und Funktionstest des Lokalisationsalgorithmus wurde die Unsicherheit zum Beispiel zeitweise entfernt.

### 4.3.1 Messwerte der Encoder

Wie beim Robotermodell schon beschrieben, wird in der Simulation dessen X/Y-Koordinate sowie der Winkel zur X-Achse als Repräsentation der Pose verwendet.



Aus den Positions- ( $\Delta s$ ) und Orientierungsänderungen ( $\Delta\psi$ ), in jedem Simulationsschritt, werden Drehwinkeländerungen ( $\Delta\alpha_{r/l}$ ) der beiden Räder berechnet:

$$\Delta\alpha_{rechts} = \frac{\Delta s + \Delta\psi \cdot D_r}{R_r} \quad (4.1)$$

$$\Delta\alpha_{links} = \frac{\Delta s - \Delta\psi \cdot D_l}{R_l} \quad (4.2)$$

mit Abstand  $D_{r/l}$  des Rades von der Mitte der Achse und Radradius  $R_{rad}$ . Die Drehwinkeländerungen werden aufsummiert um die Radstellungen zu speichern:

$$\alpha = \alpha + \Delta\alpha \quad (4.3)$$

Aus diesen Winkelstellungen<sup>4</sup> der Räder wird ein Wert für die Encoder abgeleitet und auf ganze Impulse gerundet:

$$I_{rechts} = \alpha_{rechts} \cdot \frac{g \gamma}{2\pi} \quad (4.4)$$

$$I_{links} = \alpha_{links} \cdot \frac{g \gamma}{2\pi} \quad (4.5)$$

mit Getriebeübersetzung:  $g$  und Encoder-Auflösung(pro Umdrehung):  $\gamma$

### 4.3.2 Messabweichung der Encoder

Es können drei verschiedene Typen von Messabweichungen simuliert werden:

Eine **systematische Abweichung** bei der Umrechnung vom Radwinkel in Encoderimpulse. Dies führt zu einem sich akkumulierenden Fehler in der Gefahrenen Strecke und der Orientierung. In der Simulation wurde diese Art Unsicherheit durch einen veränderten Radradius erzeugt. Für beide Räder wird vor jedem Simulationslauf in einem Fehlerintervall der Radius leicht verändert. Je stärker die Radien sich unterscheiden, umso stärker scheint der Roboter bei geradeaus Fahrt zu einer Seite zu drift. Diese Art Messabweichung ist also über die Dauer eine Simulation konstant, aber für jeden Simulationslauf zufällig.

Eine **systematische Abweichung** die nur bei Drehungen auftritt. Dies ist typischerweise die dominante Fehlerquelle bei radgetriebenen Robotern. Bei einer Drehung ist der Auflagepunkt der Räder nicht bekannt, da der innen liegende Teil des Rades sich langsamer drehen müsste als der äußere. Dies führt bei einer Drehung dazu, dass der Abstand des Rades zur Drehachse nicht genau bekannt ist. In der Simulation wird daher dieser Abstand über die Breite der Räder zufällig bestimmt. Realistischer weise müsste dieser Abstand für jede Drehung neu bestimmt werden.

<sup>4</sup>Die Winkelstellung der Räder wird nur numerisch simuliert und ist am Modell nicht sichtbar.

Zum testen der Lokalisation reicht es aber aus, wenn dies nur einmal zu Beginn einer Simulation geschieht. Dadurch treten bei Drehungen Messabweichungen auf, die zwar über einen Simulationslauf konstant und damit systematisch sind, aber die Lokalisierung genau so erschweren wie zufällige Abstände während der Laufzeit.

Eine **zufällige Abweichung** bei den Encoder-Messwerten soll ein mögliches Spiel in Kupplung oder Getriebe simulieren. Dabei wird zu dem akkumulierten Encoder-Wert eines Rades ein Gaußsches Rauschen addiert, welches proportional zur letzten Winkeländerung ist. Ist diese bei Stillstand gleich Null, so wird nur ein schwaches Grundrauschen addiert.

### 4.3.3 Bilder der Kamera

Die Kamera des Roboters wird in der Simulation nicht als Modell angezeigt. Sondern lediglich durch ein Objekt der Klasse `osgViewer::View` implementiert. Um einer echten Kamera möglichst nah zu kommen, wurde die Auflösung sowie der Öffnungswinkel der am Roboter verbauten Kamera in der Simulation nachempfunden. Die Auflösung der Bilder ist mit 1280 x 720 Pixeln im 16:9 Format. Der Horizontale Öffnungswinkel beträgt 74 Grad. Die verwendete OSG Bibliothek setzt auf OpenGL auf und nutzt deshalb auch die dort verwendeten Funktionen zur Perspektivischen Projektion. In Abbildung 4.3 ist zu sehen, wie diese in OpenGL beschrieben wird. Ein Pyramiden Stumpf begrenzt dabei ein Volumen, in dem alle sichtbaren Objekte liegen. Alles, dass sich außerhalb dieses Bereiches befindet ist im Bild nicht zu sehen. Dieser Pyramiden Stumpf wird auch **frustum** genannt, und seine Eigenschaften werden üblicher Weise in einer Projektionsmatrix zusammen gefasst. Diese lässt sich direkt setzen, oder über eine Hilfsfunktion durch folgende vier Parameter definieren:

**fovy** Gibt den vertikalen Öffnungswinkel in Grad an.

**aspect** Gibt das Verhältnis zwischen horizontalem und vertikalem Öffnungswinkel an. Er sollte gleich dem Seitenverhältnis der Bildauflösung gewählt werden, um keine Verzerrungen zu bekommen. (Breite/Höhe)

**zNear** Gibt den Abstand zwischen Betrachter und Projektionsfläche an. Sie wird auch zum clipping verwendet. (immer positiv)

**zFar** Gibt den Abstand zur hinteren clipping Ebene an. Alles was weiter als diese Ebene entfernt ist, wird nicht mit auf das Bild projiziert. (immer positiv)

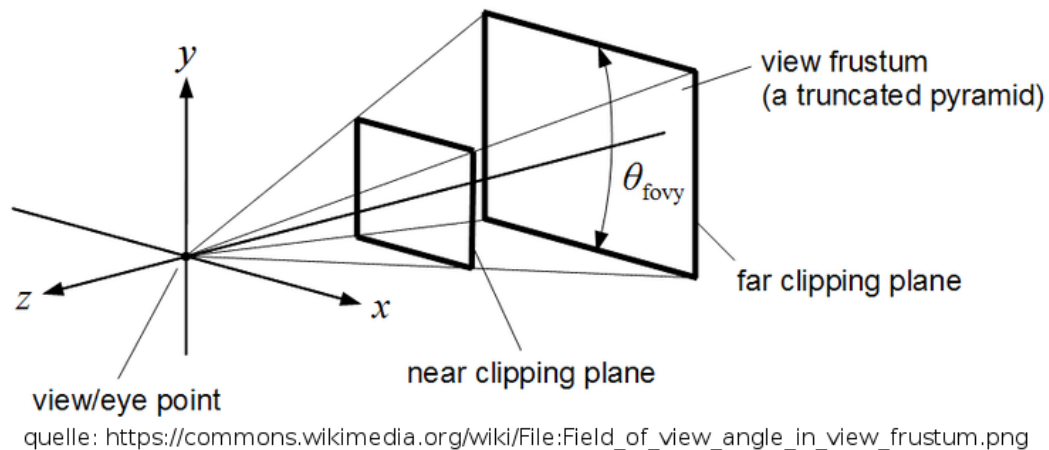


Abbildung 4.3: OpenGL beschreibt die perspektivische Projektion mit Hilfe eines **frustum**

OpenGL projiziert zunächst auf eine normalisierte Bildebene. Aus dieser normalisierten Darstellung muss mittels einer Viewporttransformation in die Bildschirmkoordinaten (Pixel) umgerechnet werden. Der Viewport ist dabei der Bildschirmbereich, auf den das fertig gerenderte Bild angezeigt wird. Mit OSG lässt sich der Viewport für eine Kamera einfach mit der gewünschten Auflösung (Höhe, Breite) und einem Ursprung setzten. Die Matrix für die Viewporttransformation heißt in OSG `windowMatrix`, und wird automatisch aus dem gesetzten Viewport berechnet.

Die Bilder der Roboter Kamera werden aber nicht auf dem Bildschirm angezeigt, sondern in einen Buffer geschrieben und von dort in ein für OpenCV nutzbares Format kopiert. In dieser Form werden die Bilder auf der Festplatte gespeichert und an die Lokalisation als Messupdate gegeben.



## 5 Versuche in der Simulation

Die Versuche in diesem Kapitel sollen helfen das entwickelte Verfahren zu bewerten. Es wird einen Versuch geben, um das Messmodell des Partikel-Filters zu untersuchen. Danach wird anhand von verschiedenen Fahrkurven das Verhalten bei unterschiedlichen Bildaufnahmefrequenzen betrachtet, sowie der Einfluss von Personen auf der Bühne beurteilt.

### 5.1 Versuche zur Genauigkeit des Messmodells

In einer ersten Versuchsreihe, soll die Genauigkeit des im Partikelfilter verwendeten Messmodells untersucht werden. Dafür soll zu einem gegebenen Bild möglichst der gesamte Zustandsraum mit dem in Kapitel 3.2.3 beschriebenen Verfahren bewertet werden. Die Anzahl der Punkte im Zustandsraum, die einen hohen Score erhalten, lässt darauf schließen wie genau die Zuordnung zwischen Bild und Pose funktioniert. Im ersten Durchlauf soll erst mal grob der gesamte Raum abgesucht werden. Er soll einen Überblick geben, wo starke Antworten liegen, die dann in weiteren Durchläufen genauer abgetastet werden können.

Für die Versuchsreihe wird der Zustandsraum zunächst grob in 5 cm Rasterschritten durchsucht. Dabei wird die Orientierung in 0,05 Rad ( $2,86^\circ$ ) Schritte Aufgeteilt. In drei Schleifen wird so der Zustandsraum durchlaufen. Als Messung wurde ein am Ursprung ( $x = 0$ ,  $y = 0$ ,  $\psi = 0$ ) aufgenommenes Bild verwendet. Für die Darstellung in Abbildung 5.1 oben, wurden Höhenlinien der Messantwort geplottet. Dabei wurde jeweils nur die stärkste Antwort an einer Position über alle Winkel gewählt. Bei einem Zweiten Durchlauf wurde die Positionsraasterung auf 1 cm Schritte, und die Winkelschritte auf 0,005 Rad ( $0,28^\circ$ ) verkleinert. Zu sehen ist dies auf Abbildung 5.1 unten. Um den Einfluss des Winkels darzustellen wurde in Abbildung 5.2 die Antwort entlang einer Linie mit  $x = 0.0$  geplottet.

In Abbildung 5.1 oben, kann man sehen, das die Antwort des Messmodells nur um die erwartete Position sichtbar einen hohen Score ergibt. Darunter sieht man

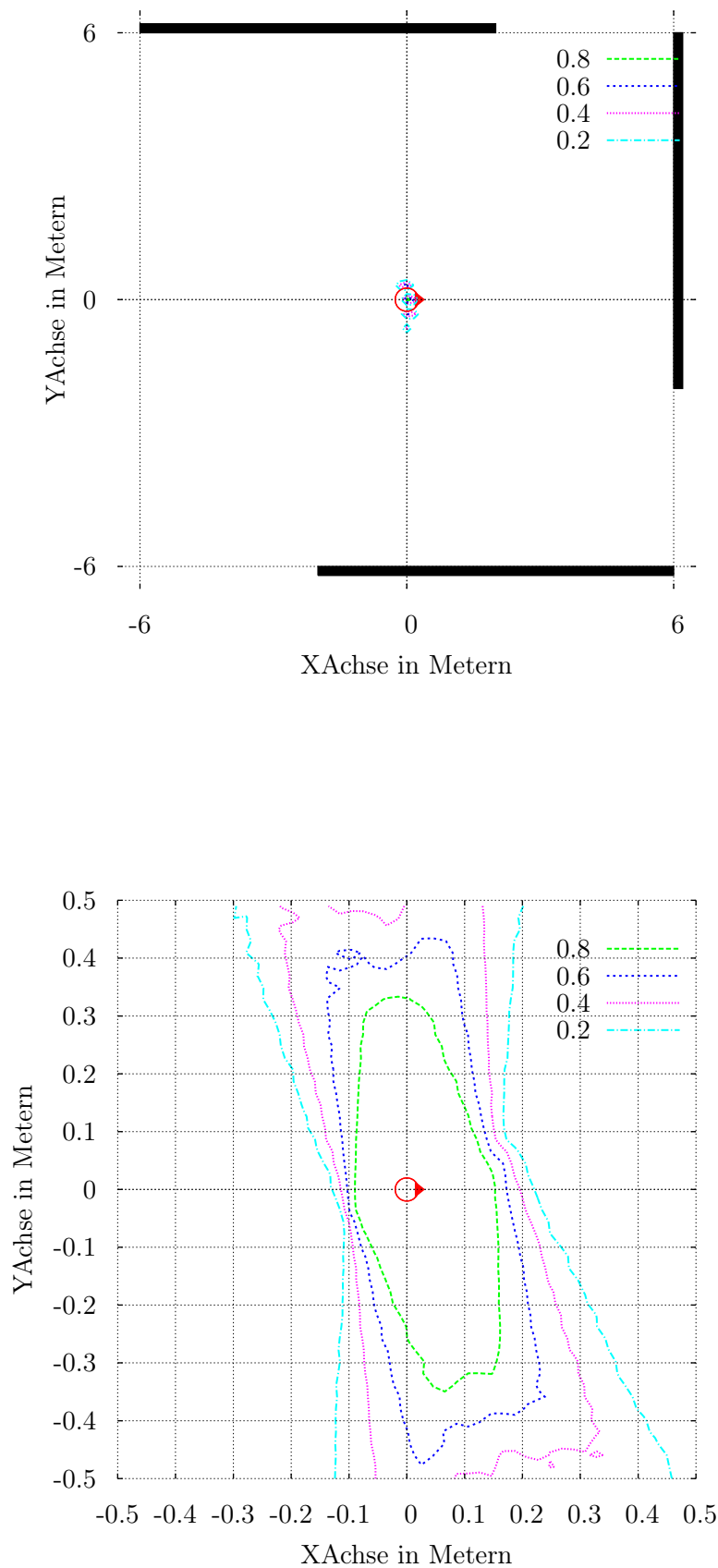
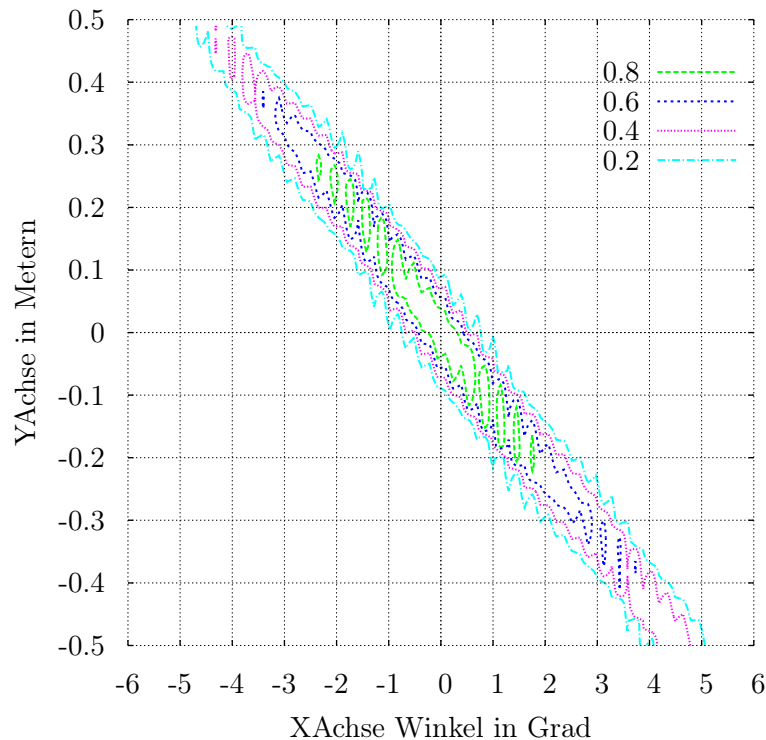


Abbildung 5.1: Antwort des Messmodells zu einem Bild an der rot markierten Position  
oben: 5cm Raster 3° Schritte unten: 1cm Raster 0.3° Schritte

Abbildung 5.2: Antwort des Messmodells bei  $x=0.0$ 

einen Vergrößerten Ausschnitt. Hier fällt auf, dass die Häufung sich leicht schräg liegt. Dies liegt vermutlich daran, dass die Lichtwand auf der das Muster abgebildet ist, nicht zentral vor dem Roboter steht, sondern in Y-Richtung verschoben ist. Die Häufung der Antworten streut vor allem entlang der Y-Achse. Also bei einer seitlichen Verschiebung des Roboters. Dabei sind innerhalb von  $\pm 30 \text{ cm}$  immer noch Scores oberhalb von 0,8 möglich. Entlang der X-Achse ist eine Streuung zwischen  $+15$  und  $-10 \text{ cm}$  zu erkennen. Durch die Höhenlinien sieht man, dass entlang der x-Achse der Score eine große Steigung hat. Dies führt zu einer relativ scharfen Kante in der in dieser Richtung. Auf der y-Achse dagegen, ist die Steigung schwächer und bildet keine so scharfe Kante. Aus Abbildung 5.2 lässt sich entnehmen, dass bei einer Verschiebung entlang der Y-Achse eine Drehung des Roboters nötig ist, um eine Pose zu erhalten die einen guten Score erhält.

## 5.2 Verwendete Fahrkurven

Es sind vier verschiedene Fahrkurven für die folgenden zwei Versuche verwendet worden. An dieser Stelle soll kurz auf Besonderheiten hingewiesen werden, die für diese Kurven beobachtet wurden. Für alle Kurven gilt, dass sie in der Mitte der Bühne beginnen, und die Lokalisation mit der Startposition initialisiert ist. Das bedeutet, dass die Versuche sich auf das Pose Tracking der Lokalisation beziehen. Auf den Abbildungen 5.3 und 5.4 sind die Lichtwände zur Orientierung als Schwarze Rechtecke in die Diagramme eingezeichnet worden.

**Loop Anticlockwise** Ist eine Fahrt entlang der äußeren Kanten der Bühne, dabei fährt der Roboter gegen den Uhrzeigersinn. Die Sichtbarkeit der Lichtwände ist bei dieser Richtung etwas schlechter, da die Wände nicht auf der Ganzen Länge der Bühne stehen, sondern immer versetzt. Der Roboter fährt immer auf die offene Stelle einer Bühnenseite zu. Es müssten weniger Bit-Musterpunkte im Bild zu sehen sein, als bei der Fahrt im Uhrzeigersinn. Aber genug um einen Score zu bilden. Zu sehen auf Abbildung 5.3 oben auf Seite 39. Es ist zu erkennen, dass auch entlang der offenen Seite die Unsicherheit groß bleibt. Obwohl auf eine der Lichtwände zu gefahren wird. Dies liegt vermutlich an der Versetzt aufgestellten Wand. Der Roboter fährt hier auf ein offenes Stück zu und die Mess-Updates reichen nicht um die Position genauer eingrenzen zu können.

**Loop Clockwise** Diese Fahrt ist der ersten sehr ähnlich, bis auf die Fahrtrichtung. Diese ist im Uhrzeigersinn. Der Roboter hat die meiste Zeit mindestens eine Lichtwand im Bild. Nur wenn er auf die Offene Seite zu fährt erhält er kein auswertbares Messupdate. Zu sehen auf Abbildung 5.3 unten auf Seite 39. Die Unsicherheit der Position nimmt mit auf dem Stück zur offenen Seite hin deutlich zu. Aber sobald wieder eine Lichtwand ins Bild kommt, nimmt sie wieder ab.



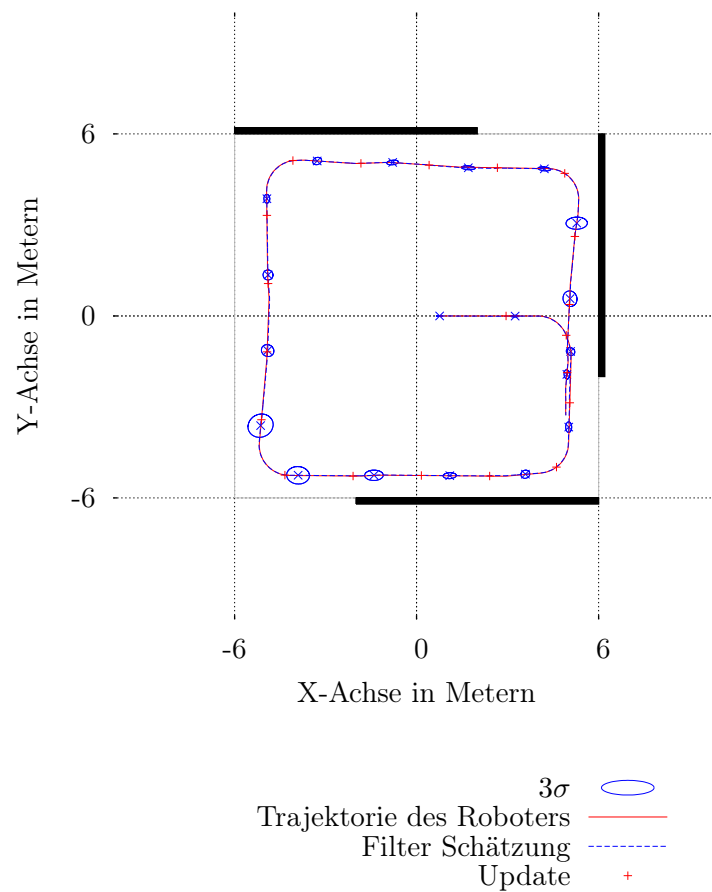
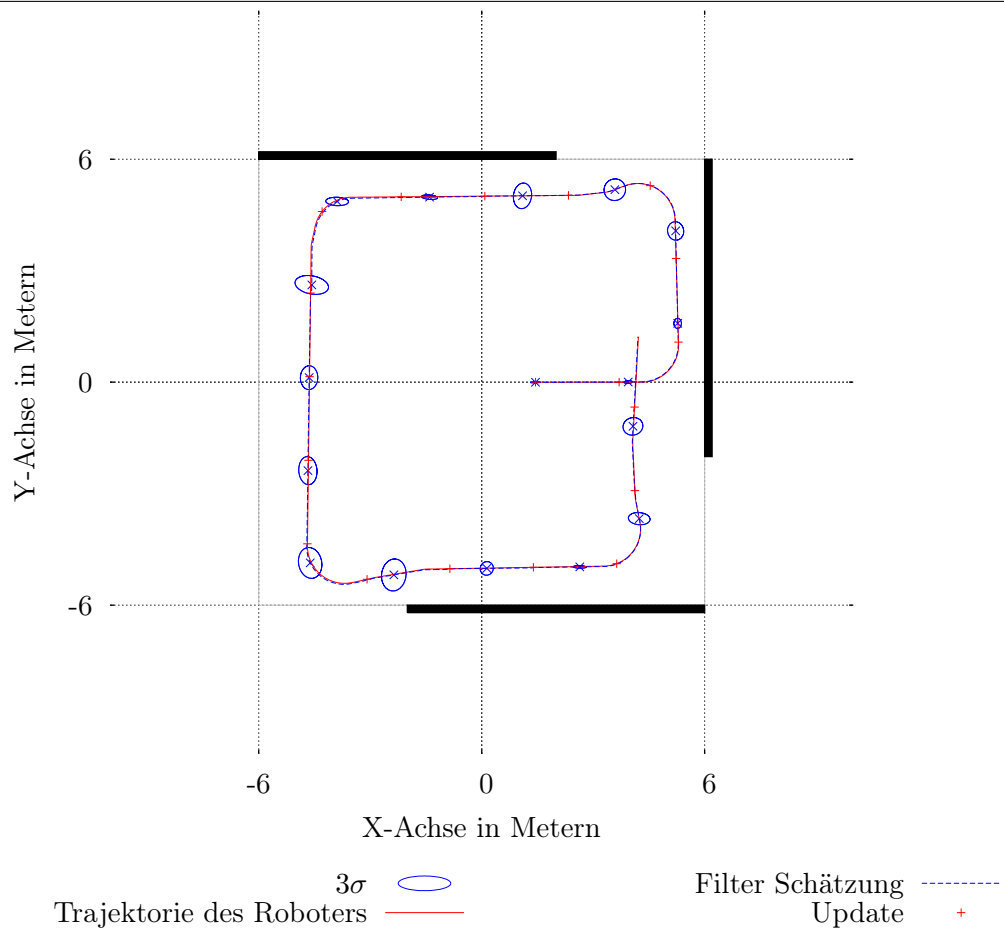


Abbildung 5.3: Fahrkurven oben gegen, und unten im Uhrzeigersinn

**Schlangenlinien** Bei dieser Fahrt fährt der Roboter mehrere male zwischen der Seite ohne Lichtwand und der ihr gegenüber liegenden Lichtwand hin und her. Dabei fährt er in Schlangenlinien die Bühne ab. In dieser Fahrkurve sind besonders viele scharfe Kurve enthalten. Außerdem wird immer wieder auf die offene Seite zu gefahren, so dass es zeitweise keine auswertbaren Bilder gibt. Zu sehen auf Abbildung 5.4 oben auf Seite 41. Besonders stark wird die Unsicherheit wenn der Roboter oben links im Diagramm dicht an der Lichtwand entlang und auf die offene Seite zu fährt. Außerdem ist gut zu sehen, wie die Unsicherheit größer wird, wenn der Roboter in negativer x-Richtung auf die offene Seite zu fährt und wieder kleiner wird, wenn er in positiver x-Richtung auf die Lichtwand zu fährt.

**Ellipse** Eine schräge Ellipse die um den Mittelpunkt liegt. Dabei fährt der Roboter mal auf zwei Lichtwände zu und mal auf die Ecke der Bühne in der keine Lichtwände stehe. Während er auf die offene Seite zu fährt, wird er zeitweise keine Lichtwand sehen um einen verwertbare Messung zu erhalten. Zu sehen auf Abbildung 5.4 unten auf Seite 41. Entsprechend verhält sich die Unsicherheit auf diesem Abschnitt. Sie wird größer, bis die Kamera wieder ein Bild mit Lichtwand liefert. Auf der Fahrt auf die Lichtwände zu bleibt die Unsicherheit klein.

### 5.3 Versuche zur Bildaufnahmefrequenz

Es soll untersucht werden in welchen Intervallen Bilder benötigt werden. Wie stark der Einfluss der Fahrkurve darauf ist, und welche Faktoren die Bildaufnahmefrequenz beeinflussen können. Dazu werden die vier beschriebenen Fahrkurven eingesetzt. Sie werden in einem Simulationslauf dreimal abgefahren. Dabei wird der Abstand zwischen der wahren und der vom Filter geschätzten Position des Roboters errechnet und über die Simulationsdauer ein Mittelwert  $\hat{d}$  bestimmt.

$$\hat{d} = \frac{1}{n} \sum_{i=0}^n \sqrt{(x_i - x_{true_i})^2 + (y_i - y_{true_i})^2} \quad (5.1)$$

Da jede Simulation mit zufälligen systematischen Messabweichungen arbeitet, werden zehn Simulationsdurchläufe bei gleichen Einstellungen durchgeführt und über deren Ergebnisse gemittelt. Dann wird die Bildaufnahmefrequenz gesenkt, bzw. die Zeit zwischen zwei Bildaufnahmen vergrößert. Als Schritte wurden 500, 1000, 2000 und 4000 *ms* gewählt. Für jeden dieser Schritte wurde jede Fahrkurve also 10 mal

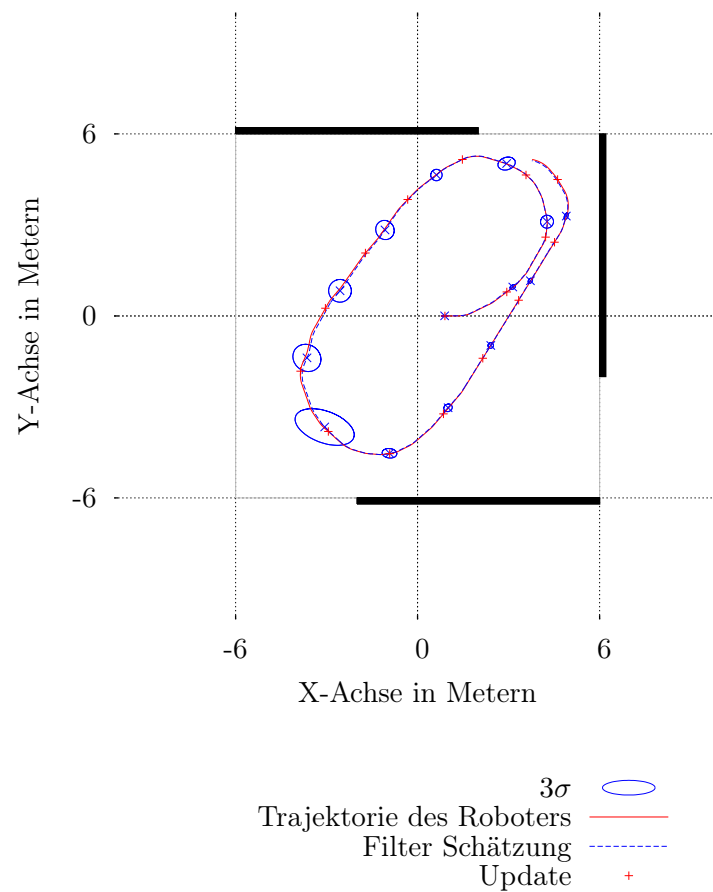
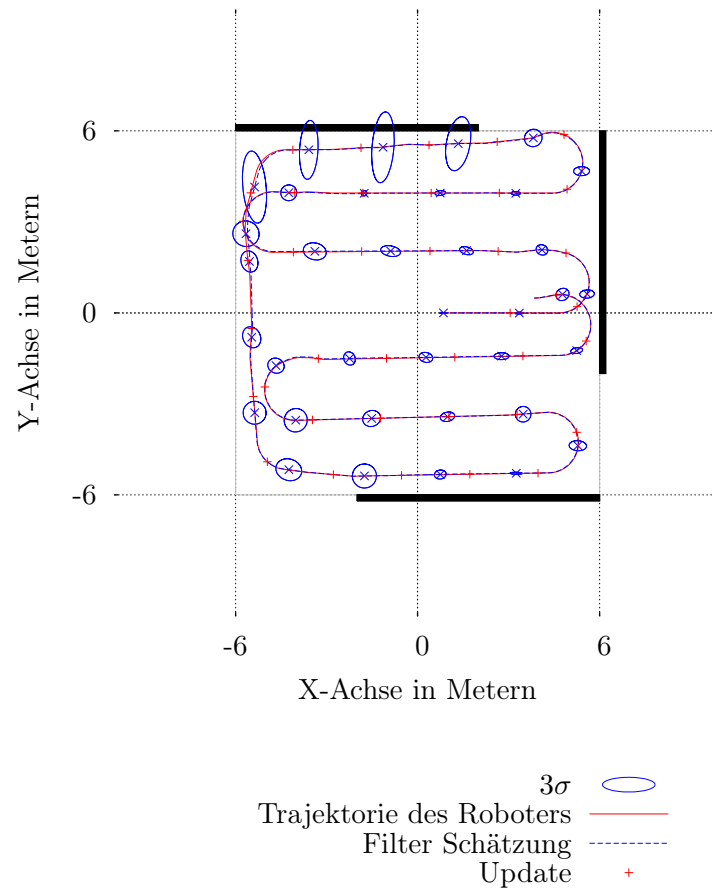


Abbildung 5.4: Fahrkurven oben Schlangenlinien, unten schräge Ellipse

simuliert. Die Ergebnisse sind in Abbildung 5.7 zu sehen. **TODO ergebnisse beschreiben und schlussfolgerungen ziehen**

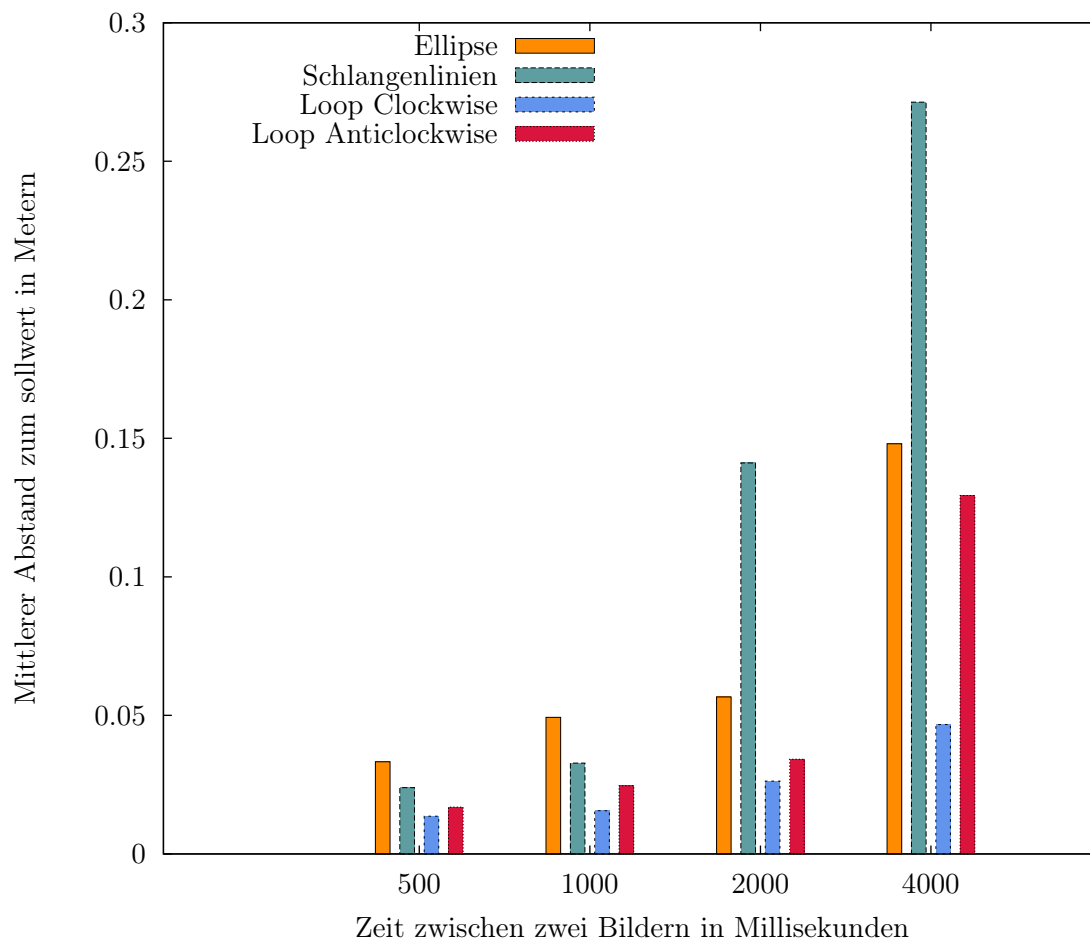


Abbildung 5.5: Gemittelter Abstand von der Fahrkurve über verschiedene Bildaufnahme-frequenzen mit vier verschiedenen Fahrkurven.

## 5.4 Versuche zum Einfluss von Menschen auf der Bühne

Hier sollen Versuche zeigen, wie stark Menschen auf der Bühne die Lokalisation mit dem hier vorgestellten Verfahren stören. Dabei geht es im wesentlichen darum, dass Menschen im Bild stehen und so die Bit-Muster ganz oder teilweise verdecken. Dazu werden ebenfalls die in Abschnitt 5.2 beschriebenen Fahrkurven verwendet. Genau wie bei den Versuchen zur Bildaufnahme-frequenz wird auch hier der mittlere Abstand zur wahren Roboterposition verwendet. Die Durchführung ist die gleiche wie zuvor, nur dass als Zeit zwischen zwei Bildern auf 1500 *ms* eingestellt wurde. Statt Bildaufnahme-frequenz wird nun die Anzahl der Personen auf der Bühne von

3 auf 9, 15 und 21 erhöht.

Für die Positionen der Personen auf der Bühne gibt es 23 festgelegte Stellen. Aus diesen wird bei jedem Simulationslauf zufällig die benötigte Anzahl gezogen. Auf Abbildung 5.6 sieht man die möglichen Positionen der Personen. Wie in Abschnitt 4.2 beschrieben sind die Personen 1800 mm hoch und 400 mm breit. Die Vermutung wäre, je mehr Personen sich auf der Bühne befinden, umso häufiger kommt es vor, dass sie im Bild stehen und das Bit-Muster teilweise oder ganz verdecken. Also müsste bei einer hohen Personendichte der mittlere Abstand von der wahren Position größer werden.

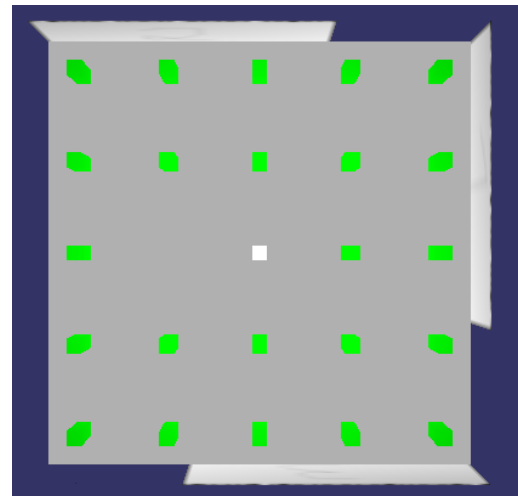


Abbildung 5.6: Mögliche Positionen von Personen

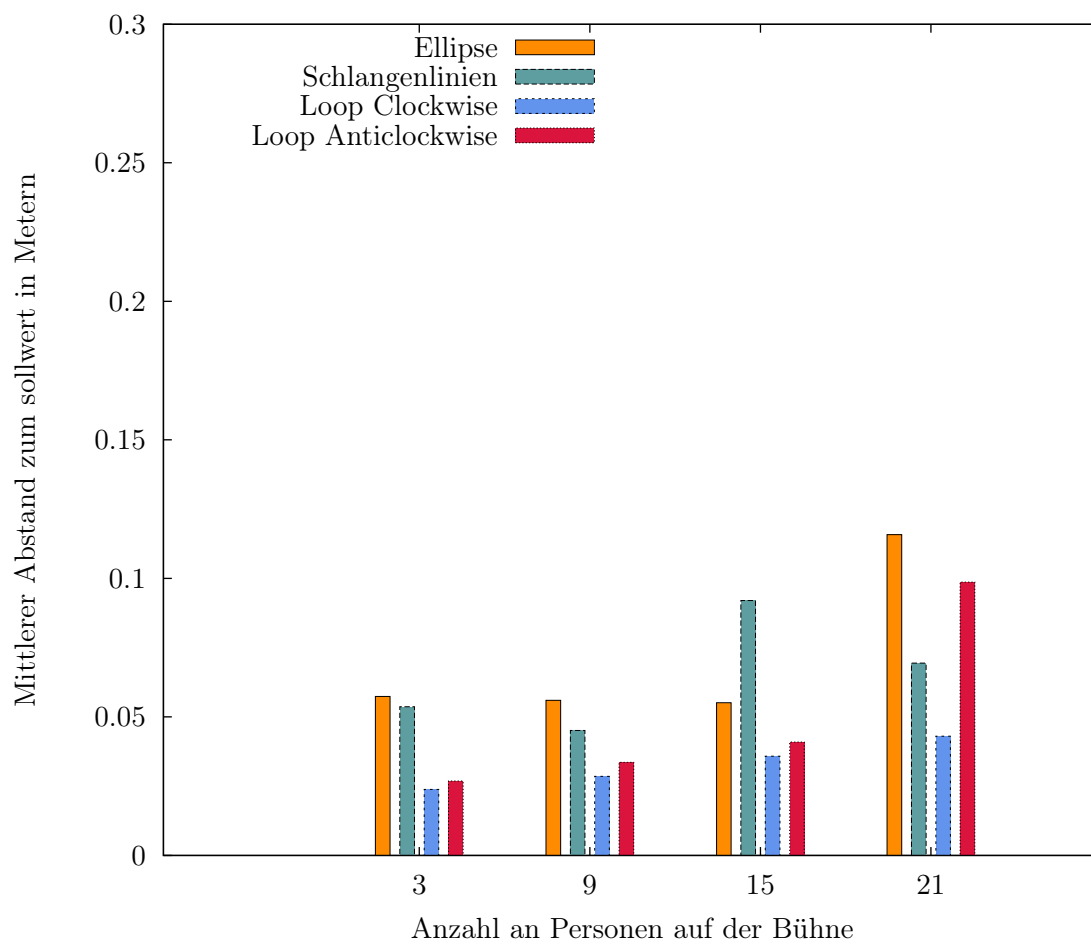


Abbildung 5.7: toller text

## 6 Fazit und Ausblick

## A Abkürzungsverzeichnis

**CDT** *C/C++ Development Tools*

**OSG** *Open Scene Graph*

**BBM** *Beobachter der Bediener von Maschinen*

**SLAM** *Simultaneous Localization and Mapping*

**MCL** *Monte Carlo Localization*





## B Literaturverzeichnis

- [Ha et al., 2012] Ha, X. V., Ha, C., and Lee, J. (2012). *Intelligent Computing Technology: 8th International Conference, ICIC 2012, Huangshan, China, July 25-29, 2012. Proceedings: Trajectory Estimation of a Tracked Mobile Robot Using the Sigma-Point Kalman Filter with an IMU and Optical Encoder*, volume 7389 of *Lecture Notes in Computer Science*. Berlin, Heidelberg.
- [Hertzberg et al., 2012] Hertzberg, J., Lingemann, K., and Nüchter, A. (2012). *Mobile Roboter: eine Einführung aus Sicht der Informatik*. eXamen.press. Springer Vieweg, Berlin [u.a.]. X, 389 S. : zahlr. Ill. und graph. Darst.
- [Jähne, 2005] Jähne, B. (2005). *Digitale Bildverarbeitung*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 6., überarbeitete und erweiterte auflage edition. Online-Ressource : v.: digital.
- [Seco et al., 2009] Seco, F., Jiménez, A. R., Prieto, C., Roa, J., and Koutsou, K. (2009). A survey of mathematical methods for indoor localization. In *Intelligent Signal Processing, 2009. WISP 2009. IEEE International Symposium on*, pages 9–14. IEEE.
- [Thrun, 2002] Thrun, S. (2002). Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence, UAI'02*, pages 511–518, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Thrun et al., 2006] Thrun, S., Fox, D., and Burgard, W. (2006). *Probabilistic robotics*. Intelligent robotics and autonomous agents. MIT Press, Cambridge, Mass. [u.a.]. XX, 647 S. : Ill., graph. Darst.
- [Tönnies, 2005] Tönnies, K. D. (2005). *Grundlagen der Bildverarbeitung*. Informatik. Pearson Studium, München. Online-Ressource.



## C Anhang

**Code 93 bar code**

ID	Character	Widths	Binary	ID	Character	Widths	Binary
0	0	131112	100010100	28	S	211122	110101100
1	1	111213	101001000	29	T	211221	110100110
2	2	111312	101000100	30	U	221121	110010110
3	3	111411	101000010	31	V	222111	110011010
4	4	121113	100101000	32	W	112122	101101100
5	5	121212	100100100	33	X	112221	101100110
6	6	121311	100100010	34	Y	122121	100110110
7	7	111114	101010000	35	Z	123111	100111010
8	8	131211	100010010	36	-	121131	100101110
9	9	141111	100001010	37	.	311112	111010100
10	A	211113	110101000	38	SPACE	311211	111010010
11	B	211212	110100100	39	\$	321111	111001010
12	C	211311	110100010	40	/	112131	101101110
13	D	221112	110010100	41	+	113121	101110110
14	E	221211	110010010	42	%	211131	110101110
15	F	231111	110001010	43	(\$)	121221	100100110
16	G	112113	101101000	44	(%)	312111	111011010
17	H	112212	101100100	45	(/)	311121	111010110
18	I	112311	101100010	46	(+)	122211	100110010
19	J	122112	100110100	Start/Stop *		111141	101011110
20	K	132111	100011010	(Reverse stop)		114111	101111010
21	L	111123	101011000	Unused		411111	111101010
22	M	111222	101001100			111132	101011100
23	N	111321	101000110			111231	101001110
24	O	121122	100101100			113112	101110100
25	P	131121	100010110			113211	101110010
26	Q	212112	110110100			213111	110111010
27	R	212211	110110010			212121	110110110

Abbildung C.1: code 93, Quelle: Wikipedia