

Applied Research in Engineering Sciences

Abschlussbericht für Projektarbeit 1

SLAM in dynamischen Umgebungen für autonome Systeme

Dimitrij-Marian Holm

Matrikelnummer: 00512012

Hochschule für angewandte Wissenschaften München

Fakultät Elektro- und Informationstechnik

Lothstraße 64

80335, München

Email: holm0@hm.edu

INHALTSANGABE

I	Einleitung	1
I-A	Vorwort	1
I-B	Stand der Forschung	1
I-C	Problemstellung	2
I-D	Forschungsfrage & Hypothese	5
II	Mathematische und algorithmische Grundlagen	6
II-A	Grundlegende Definitionen	6
II-B	Sensorik	6
	II-B1 Kinematische Modelle	6
	II-B2 Umgebungswahrnehmung	7
II-C	Lokalisierung	9
	II-C1 Monte Carlo Lokalisierung	11
	II-C2 Adaptive Monte Carlo Lokalisierung	12
II-D	Pfadplanung	12
II-E	Kombination Lokalisierung und Pfadplanung	14
II-F	SLAM	14
	II-F1 Rao-Blackwellized Sample Mapping	15
	II-F2 GMapping	16
III	Lösungsansatz der Problemstellung	17
III-A	Voraussetzungen	17
III-B	Konzeption	18
	III-B1 Ausgangslage	18
	III-B2 Schnittstelle, Input/ Output	18
	III-B3 Lokale Umgebungsdarstellung	19
	III-B4 Analyse der Status in den Zellen	20
	III-B5 Informationen über zeitliches Verhalten der Zelle	23
	III-B6 Entscheidungsfindung	25
IV	Umsetzung & Ergebnisse	26
IV-A	Entwicklungs- und Testumgebung / Komponenten	26
V	Zusammenfassung & Ausblick	29
Referenzen		37

I. EINLEITUNG

A. Vorwort

Einer der wesentlichen Schlüsselfunktionen eines autonomen Systems ist die Erstellung einer Karte aus vorhandenen Sensorinformationen, auch SLAM genannt. Die Herausforderung in SLAM ist einerseits die Kartierung der Umgebung und auf der anderen Seite die Lokalisierung in der Karte. Die erzeugte Karte ist zudem wichtiger Bestandteil für globale Pfadplanungen. Betrachtet man die Umgebung in der das System agiert als statisch, kann das SLAM-Problem in klassischen Szenarien, z. B. in Gebäuden, als weitestgehend gelöst betrachtet werden [1] [2]. Autonome Systeme sollen jedoch auf ein langfristiges Interaktionsszenario mit der Umgebung ausgelegt sein. Die Markov-Annahme, die auf einer statischen Umgebung basiert, ist in diesem Fall nicht einhaltbar, da die Umwelt per se dynamischen Charakter hat. Der Einbezug der Dynamik in SLAM und deren Folgerung, auch langfristige Änderungen in das System einzubinden, eine Wartung der Karte, wird bereits seit einiger Zeit erforscht und ist als „SLAM in dynamischen Umgebungen“ bekannt [3] [4] [5] [6].

B. Stand der Forschung

Da sich eine Lösung des Problems „SLAM in dynamischen Umgebungen“ als wichtiger Schritt für die Funktionalität von autonomen mobilen Systemen darstellt, wurde diese bereits in einigen Projekten untersucht. Alle recherchierten Untersuchungen zu diesem Thema sehen die Art der Kartierung der Umgebung als zentralen Bestandteil für die Lösung. Prinzipiell lassen sich Konzepte der Kartierung in drei Hauptansätze aufteilen: Zum einen die metrische Karte, in der eine klassische Umgebungskarte wie in Fig. 2, auf Seite 4 vorgestellt. Auf der anderen Seite die Darstellung der Karte als Graph, das topologische Verfahren, in dem Schlüsselpositionen die Knoten bilden (Siehe Fig.1, S. 2). Im Gegensatz zum metrischen Ansatz ist eine Pfadplanung durch Techniken der Graphentheorie wesentlich einfacher, dafür sind die beinhalteten Informationen der Umgebung geringer [5]. Die Kombination beider Verfahren, topologisch und metrisch, ergibt die dritte Variante.

Als Erstes sollen einige Ansätze auf metrischer Basis vorgestellt werden:

Eine beliebte Methode ist das Aufteilen der Umgebung in einen oder mehrere dynamische Teile und einen statischen Teil. [5] [6]. So wird in [6] für jede Fahrt des Roboters durch das Areal eine Umgebungskarte kartiert, d.h. es existiert eine Vielzahl von Karten erzeugt zu unterschiedlichen Zeiten. Die Karteninformationen werden genutzt, um eine statische Karte für das gesamte Areal zu erzeugen und bewegliche Objekte zu extrahieren. Jedes gefundene dynamische Objekt erhält eine eigene lokale Umgebungskarte. Eine Aufteilung in genau eine statische und eine dynamische Karte wird in [5] beschrieben. Jeder neue Satz an Sensorinformationen führt zu einer Berechnung der Wahrscheinlichkeiten für eine Veränderung auf der statischen Umgebungskarte. Anhand der Wahrscheinlichkeiten aus dem Sensordatensatz und der statischen Karte wird die dynamische Karte berechnet. Das Verwenden einer Baumstruktur, die als Indexierung Zeitstempel verwendet, ist in [7] erläutert. Für jede Zelle in der Umgebungskarte werden Zeitindizes bereitgestellt. Jeder Index speichert die Wahrscheinlichkeit einer Belegung durch Objekte zu einer bestimmten Zeit ab. Durch die Verwertung der Zeitindizes ist eine Entscheidung über den aktuellen Status pro Zelle möglich.

Auch hier wird zwischen statischen und dynamischen Objekten differenziert. Allerdings werden bei diesen Umsetzungen die Zeit als möglicher zusätzlicher Entscheidungsfaktor nicht berücksichtigt, da er nur als Index genutzt wird. Im Gegensatz hierzu verwendet [3] die Zeit als zusätzlichen Faktor. Das gesamte Arbeitsareal wird in lokale Umgebungskarten gegliedert. Jede lokale Karte besitzt verschiedene Zeitskalierungen, d.h. es existieren mehrere skalierte Kopien der lokalen Karte. Durch die Gewichtung der Zeitskalierungen mit einer Lebensdauer, kann, abhängig des Alters der Karten, entschieden werden, welche Karte die am besten geeignete ist, um das aktuelle Areal zu repräsentieren. Die globale Karte wird bei gegebener Zeit anhand der ausgewählten lokalen skalierten Karten aktualisiert.

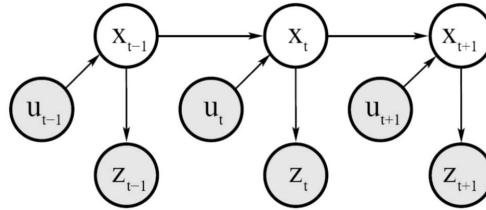


Fig. 1: Bsp. eines Graphen (Hier: dynamisches Bayes-Netzwerk), u := Steuerungsdaten= beinhalten Informationen über die Pose des Roboters, z. B. über Integration der Odometrie. x := Status= Sammlung aller wichtigen Aspekte des Roboters (Pose, Beschleunigung,...) und Umgebung. z := Umgebungsdaten= Erhalt durch z. B. Laserscan. Die Zeitpunkte werden mit Index t beschrieben, wobei $t-1 < t < t+1$, Quelle: [8]

Ein Beispiel aus der topologischen Richtung bietet [4]. Jeder Knoten in einem Standard-Graphen [4, S.4] beinhaltet die Informationen über die Pose des Roboters sowie die Daten der Sensoren. Was den Graphen in dieser Umsetzung unterscheidet, ist die Erweiterung der Knoteninformationen um eine Sektoreinteilung und Klassifizierung der Sensorinformationen. Es wird für jede Fahrt des Roboters ein solcher Graph angelegt. Um Veränderungen in der Umgebung zu erkennen, wird eine lokale Karte berechnet, basierend auf dem aktuellen Graph und vergangenen Graphen. Anhand dieser lokalen Karte werden die Informationen der Knoten aktualisiert. Auch dieser Ansatz verwertet keine Informationen über die Zeit.

Es kristallisiert sich heraus, dass viele Umsetzungen die Zeit als wichtige Informationsquelle über das Verhalten der Umwelt nicht in Betracht ziehen. Zwar werden verschiedene Informationen zu verschiedenen Zeiten genutzt, um Unterschiede zu erkennen, das Zeitintervall jedoch bleibt unberücksichtigt. Zudem kann das Differenzieren zwischen dynamischen und statischen Umgebungen beim Eintreten gewisser Szenarien, wie z. B. das Entfernen von Objekten, die sehr lange statisch waren, die Anpassungsfähigkeit der Umgebungskarte nachteilig beeinflussen.

C. Problemstellung

Um das in dem Forschungsprojekt zu behandelnde Problem besser darstellen zu können, werden die Techniken der lokalen und globalen Wegplanung erläutert. Die lokale Wegplanung nutzt die aktuellen Informationen aus den Sensoren des Roboters (z. B. Kamera, Ultraschall, Laserscan). Aus der Menge

der aus der Umwelt extrahierten Daten wird ein lokales Abbild der Umgebung erstellt. Die maximale Reichweite der Sensoren definiert folglich die Sichtweite des Roboters und begrenzt die Reichweite der planbaren Trajektorie. Ein wesentlicher Vorteil dieser Wegplanung ist die Möglichkeit, auf Objekte und lokale Veränderungen in der Umgebung innerhalb der Sichtweite des Roboters zu reagieren. Ein schwerwiegender Nachteil ist das Fehlen einer Planung, die über die Grenzen der Sichtweite eine Trajektorie berechnen kann. Zur Veranschaulichung wird die Wahrnehmung, Orientierung und Wegplanung eines Menschen als Analogie herangezogen. Befindet man sich in einem unbekannten Raum, so kann man sich nur anhand von den menschlichen Sensoren, Sinnesorganen (Augen, Ohren, ...) orientieren, erkennt Objekte und nimmt somit den Raum wahr. Was sich hinter verdeckenden Objekten befindet ist nicht erkennbar. Es ist erst möglich, seine Planung anzupassen, sobald verdeckte Objekte durch eine Bewegung des Menschen oder des Objekts in den Aktionsbereich der menschlichen Sensoren gerät. Eine sinnvolle Planung über die Reichweite der Sinnesorgane hinaus ist aber dennoch nicht möglich. Um diese Einschränkung zu umgehen, ist eine globale Wegplanung vonnöten.

Um längere Trajektorien berechnen zu können, wird auf die globale Wegplanung zurückgegriffen. Diese benötigt als Grundlage eine Umgebungskarte von dem Areal, auf dem der Roboter autonom agieren soll. D. h., der Roboter befand sich bereits im Arbeitsbereich und konnte durch eine Exploration eine Karte der Umgebung erstellen oder es existiert bereits eine Karte, z. B. ein angepasster Gebäudeplan. Erstellt der Roboter seine Karte selber, wird dies in den meisten Anwendungen durch einen SLAM-Algorithmus umgesetzt [8, S.309 ff.]. Ein Ausschnitt einer Umgebungskarte, wie sie in Robotersystemen verwendet wird, ist in Fig. 2, S. 4 gezeigt. Da solchen Karten oft nur keine Aktualisierungsintervalle widerfahren, ist diese Informationsquelle als statisch zu betrachten. Mit Hilfe der Umgebungskarte hat der Roboter Informationen über das gesamte Arbeitsareal und kann Trajektorien zwischen Zielen berechnen, die sich außerhalb der Sensorreichweite befinden und somit nicht mittels lokaler Planung berechenbar wären. Für ein autonom mobiles System ist diese Funktion essenziell. Auch hier ist eine Analogie zum Menschen sinnvoll. Wurde ein Gebiet bereits erkundet, speichert der Mensch die Informationen seiner Umgebung in seinem Gedächtnis ab. Somit können Wege auch über große Distanzen, z. B. über mehrere Räume geplant werden.

Fährt der Roboter nun an seiner geplanten Bahn entlang und ein Objekt, das nicht auf der Umgebungskarte verzeichnet ist, befindet sich genau auf der geplanten Wegstrecke, wird es unweigerlich zu einer Kollision kommen. Der Planer hat durch den Mangel an Informationen die Trajektorie durch das blockierende Objekt geplant, die dem globalen Planer jedoch nicht bekannt ist. Aufgrund der Verwertung von einer statischen Informationsquelle ist es der globalen Planung nicht möglich, auf solche Szenarien zu reagieren. Zur Analogie: Würde man der Person anschließend die Sinnesorgane dämpfen, beispielsweise durch Verbinden der Augen, eine Kollision mit Objekten, die nicht im Gedächtnis eingeprägt sind, wäre auf dem Weg zur Zielposition unvermeidlich. Die Kombination beider Planungsarten ergibt eine stabile Gesamtplanung. Während der globale Planer die Wegfindung zwischen dem Start- und Endpunkt berechnet, fließt die Abbildung der unmittelbaren Umgebung über den lokalen Planer in die Berechnung ein und kann somit auf lokale Änderungen

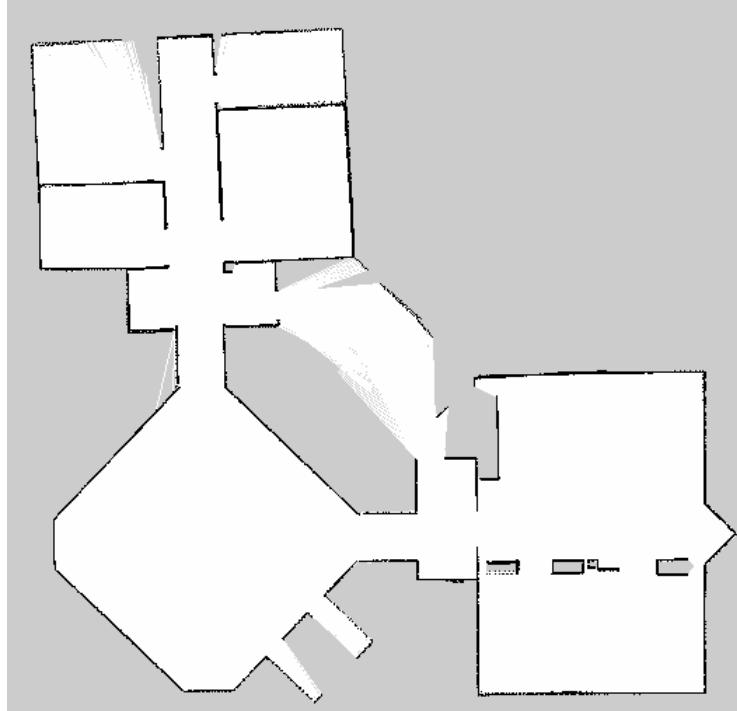


Fig. 2: Beispiel einer Umgebungskarte nach einer Exploration mit SLAM. Weiß entspricht Zellen, die den Status Frei besitzen, Schwarz Besetzt und Grau sind Zellen ohne Informationen über den Status

reagieren.

Um eine konkrete Vorstellung der auftretenden Probleme in der Pfadplanung innerhalb dynamischer Umgebungen zu erhalten, wird folgendes Szenario beschrieben:

Während der Fahrt von einem Startpunkt zu einem Zielpunkt über eine längere Strecke (geplante Wegstrecke ist wesentlich größer als die maximale Sensorreichweite), taucht ein Objekt auf der geplanten Trajektorie auf, das nicht auf der Karte markiert ist. Es wird davon ausgegangen, dass sich der Ort des Objekt während dem gesamten Szenario nicht verändert. Sobald das Objekt innerhalb der Sensorreichweite ist, reagiert der lokale Planer und wird versuchen, einen Weg um das Objekt zu planen, der global geplante Streckenverlauf wird kurzzeitig verlassen. Der lokale Planer wird versuchen, sich so schnell wie möglich wieder an dem global geplanten Weg zu orientieren. Wenn der Roboter diese Strecke nun mehrfach von A nach B und wieder zurück fährt, könnte es sich als sinnvoll erweisen, das Objekt in die globale Wegplanung einzubeziehen, sprich das Objekt in der Umgebungskarte einzutragen.

Das Gleiche gilt auch für Objekte, die auf der Umgebungskarte eingezeichnet, aber in der realen Umgebung nicht mehr vorhanden sind. In diesem Fall berechnet der globale Planer einen Weg um das Objekt. Sobald die Reichweite der Sensoren das Objekt laut der Umgebungskarte erkennen sollten, dies aber nicht können, wird der lokale Planer einen Weg durch den besetzten Bereich auf der

Umgebungskarte planen.

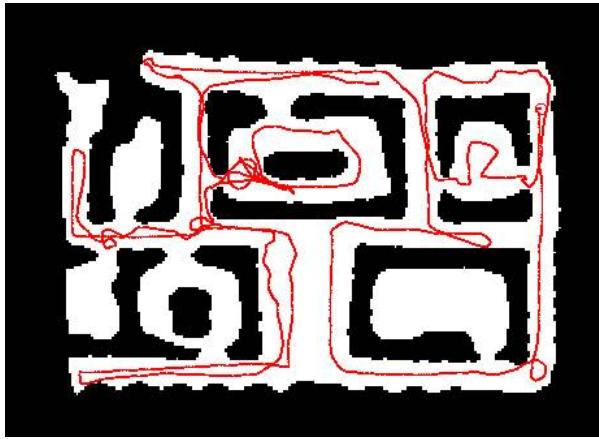


Fig. 3: Realer Pfad eines Roboters durch seine Umgebung. Quelle: [8]

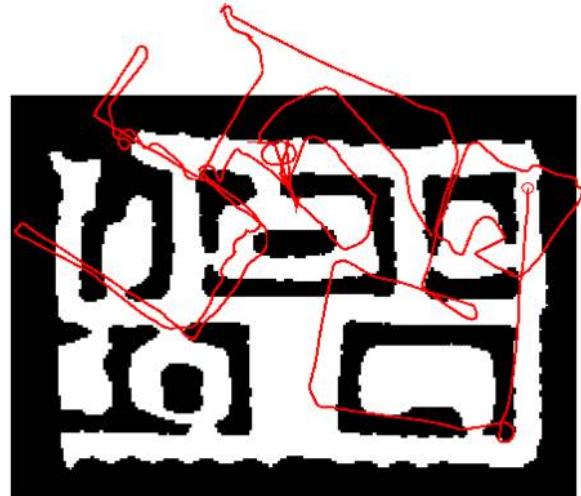


Fig. 4: Pfad, aufgezeichnet anhand der Odometriedaten. Quelle: [8]

Ein zusätzliches Problem kommt aus dem Bereich der Lokalisierung. Der Roboter hat hierbei die Aufgabe, seine eigene Pose in der Umgebung, z. B. auf der Umgebungskarte, zu ermitteln. Wird von einer bekannten Startposition ausgegangen, spricht man von einer lokalen Lokalisierung. Um von Startpunkt aus eine Veränderung der Pose bestimmen zu können, werden diverse propriozeptive Sensoren eingesetzt, hauptsächlich Inertial Measurement Units (IMU) und Odometrie. Kein Sensor ist jedoch fehlerfrei, es können unter anderem Rauschen und Biassse auftreten oder externe physikalische Gegebenheiten, wie z. B. Schlupf an den Rädern. Dies führt zu Fehlern in der Bestimmung der Pose. In Fig. 4 auf S. 5 ist der Pfad eines Roboters nur anhand der vorhandenen Odometriedaten rekonstruiert. Es ist sehr gut erkennbar, dass sich der Sensorfehler aufsummieren und die Diskrepanz im Verhältnis zu dem wirklich zurückgelegten Pfad in Fig. 3 , S. 5 kontinuierlich steigt. Um die Lokalisierungsgenauigkeit zu erhöhen werden für die Posenbestimmung zusätzlich exterozeptive Sensoren, beispielsweise Laserscanner, Kameras und Ultraschall. Diese Informationen werden mit einer Umgebungskarte korreliert. Unstimmigkeiten zwischen der Umgebungskarte und den Sensorinformationen, beispielsweise physisch vorhandene Objekte, die nicht in der Umgebungskarte eingetragen sind, können die Genauigkeit der Lokalisierung reduzieren.

D. Forschungsfrage & Hypothese

Autonome Systeme besitzen viele Aufgabenbereiche, in denen eine möglichst genaue Umgebungskarte benötigt wird. Die Umwelt kann sich jedoch kontinuierlich ändern. Aufgrund der beschriebenen Problematik stellt sich die Frage, ob es Möglichkeiten gibt, die Dynamik der Umwelt sinnvoll in der Umgebungskarte einzubinden, die Karte zu modifizieren, und es somit dem Roboter ermöglicht,

auf Änderungen im Zielareal langfristig zu reagieren. Ein erster Ansatz ist, neben der Betrachtung der Objekte in Verbindung mit der Umgebungskarte, also Ortseigenschaften (Besetzt, Frei), zusätzlich die Zeit als Entscheidungsfaktor einzubeziehen und anhand eines Entscheidungsalgorithmus eine Modifikation der Umgebungskarte durchzuführen. Dies ist ein grundlegender Unterschied zu anderen Ansätzen, wie in I-B, S. 1 geschildert. Ein zusätzlicher Gesichtspunkt ist die Betrachtung der Umwelt als komplett dynamisch. Jedes in der Realität persistente Objekt erhält somit eine volatile Eigenschaft. So ist die Flexibilität, auch auf Veränderungen zu reagieren, die nach einer sehr langen Zeit auftreten können, gegeben. Die Ziele des Forschungsprojekts sind somit die Entwicklung eines Verfahrens, das die langfristige Wartung der Umgebungskarte ermöglicht.

II. MATHEMATISCHE UND ALGORITHMISCHE GRUNDLAGEN

A. Grundlegende Definitionen

Die nachfolgend behandelten Algorithmen und mathematischen Modelle dienen zur Erklärung und Beschreibung der in IV-A auf Seite 26 gelisteten Module. Um eine Basis für die Algorithmen und mathematischen Modelle zu besitzen, werden die wichtigsten Definitionen aufgeführt:

u = Darstellung der Steuerungsdaten. Es werden die Informationen über die Änderung des Status x gesammelt. Typischerweise handelt es sich um die Daten der Geschwindigkeit oder die der Odometrie.

z = Umgebungsinformationen. Daten von Kameras, Distanzmesser (Ultraschallsensor, Laserscan), ..., zählen zu diesen Informationen.

m = Beschreibung der Umgebungskarte, siehe Fig. 2, S.4.

x = Status. Beinhaltet sämtliche wichtige Beschreibungsparameter für den Roboter und die Umgebung. Diese können sein:

- Pose des Roboters (Die Position und Orientierung relativ zu einem Weltkoordinatensystem), in Normalfall aufgeteilt in drei kartesische Variablen und drei für die Beschreibung der Winkel (Oft RPJ = Raw, Pitch, Yaw). Bei einer planaren Fläche, also bei Robotern, die in zwei Raumdimensionen agieren, reichen zwei kartesische und ein Winkelwert, meist die Blickrichtung (Yaw) zur Beschreibung.
- Parameter und Werte der Manipulatoren des Roboters, z. B. die Winkel der Aktuatoren
- Die Geschwindigkeit des Roboters, aufgeteilt in Translation und Rotation, für die Basis des Roboters in der Umgebung sowie dessen Manipulatoren.
- Die Position und Merkmale der Umgebung

Somit gilt: $u, z, m \in x$.

t = Zeitpunkt, jeder Wert (x, u, z, m) kann hiermit indexiert werden, wobei $t - k < t < t - l$ mit $k, l \in \mathbb{N}$.

B. Sensorik

1) Kinematische Modelle:

Die Grundlage für Kapitel II-C, S.9 ist die Modellierung der Kinematik des Roboters in einem mathematischen Modell. Aber auch für andere Berechnungen und Tasks sind die mathematischen

Beschreibungen essentiell. Es handelt sich nicht um die Kinematik von Manipulatoren, z. B. Roboterarme oder Greifer, sondern um die Darstellung der Fortbewegung des Roboters in der Umwelt. Es sollen die Algorithmen der Modelle aufgezeigt werden, für die Erklärung in Table I (S. 7), III (S. 8), II (S. 8) sowie deren mathematischen Herleitungen ist in [8] auf den Seiten 117 - 139 nachzuschlagen. Der erste Algorithmus beschäftigt sich mit der Darstellung der Kinematik über die Parameter der Geschwindigkeit (Table I), aufgeteilt in rotative (ω) und translativen (v) Geschwindigkeit. Die Darstellung eignet sich nicht für Mecanum-Antriebe oder Roboter mit Beinen bzw. Räumen mit mehr als zwei Dimensionen.

1 **sample_motion_model_velocity** ($u_t = \begin{pmatrix} v \\ \omega \end{pmatrix}$, $x_{t-1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$)

2 $\hat{v} = v + \text{sample}(\alpha_1 v^2 + \alpha_2 \omega^2)$

3 $\hat{\omega} = \omega + \text{sample}(\alpha_3 v^2 + \alpha_4 \omega^2)$

4 $\hat{\gamma} = \text{sample}(\alpha_5 v^2 + \alpha_6 \omega^2)$

5 $x' = x - \frac{\hat{v}}{\hat{\omega}} \sin \theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega} \Delta t)$

6 $y' = y + \frac{\hat{v}}{\hat{\omega}} \cos \theta + \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega} \Delta t)$

7 $\theta' = \theta + \hat{\omega} \Delta t + \gamma \Delta t$

8 return $x_t = (x', y', \theta')^T$

TABLE I: Die Algorithmik für die Berechnung des Status x_t über die Geschwindigkeit, wobei α_1 bis α_6 Roboter spezifische Parameter für die Fehlerabbildung in der Kinematik sind. Quelle: [8]

Eine Alternative zur Beschreibung der kinematischen Modelle ist auch über den Ansatz der Odometrie möglich (Table II). Praktische Versuche haben gezeigt, dass die Verwertung der Odometriedaten im Normalfall eine wesentlich höhere Genauigkeit im Vergleich zur Verwendung von Geschwindigkeiten aufweist. Ein Nachteil ist die Aktualität der Informationen, da diese immer in der Vergangenheit liegen. Aufgrund der besseren Genauigkeit wird in der Majorität der Anwendungen das mathematische Modell der Odometrie zur Beschreibung der Kinematik des Roboters verwendet.

2) Umgebungswahrnehmung:

Da die Odometrie, trotz besserer Genauigkeit als die Darstellung über die Geschwindigkeit, einen auf Dauer kumulativen Fehler besitzt (Siehe Fig. 4, S.5), kann ein autonomes mobiles System nicht sinnvoll ohne Informationen über die Umwelt agieren. Daher kann der Roboter mit einer großen Bandbreite an Sensoren ausgestattet werden. Dieser Fall beschreibt die Distanzmessungssensoren mit Messmedien Ultraschall und Laser. Im Gegensatz zu früher werden statt auf Sonar basierende Sensoren fast ausschließlich Laserscanner eingesetzt. Diese besitzen im Vergleich zu akustischen Sensoren wesentliche Vorteile, wie eine geringere Störanfälligkeit, höhere Aktualisierungsraten, besser fokussierte emittierte Signale und eine wesentlich bessere Auflösung, meist gerastert in 1°

```

1 sample_motion_model_odometry ( $u_t = \begin{pmatrix} \bar{x}_{t-1} = (\bar{x} \ \bar{y} \ \bar{\theta})^T \\ \bar{x}_t = (\bar{x}' \ \bar{y}' \ \bar{\theta}')^T \end{pmatrix}, x_{t-1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$ )
2  $\delta_{rot1} = atan2(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$ 
3  $\delta_{trans} = \sqrt{(\bar{y}' - \bar{y})^2 + (\bar{x}' - \bar{x})^2}$ 
4  $\delta_{rot2} = atan2(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$ 
5  $\hat{\delta}_{rot1} = \delta_{rot1} - \text{sample}(\alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2)$ 
6  $\hat{\delta}_{trans} = \delta_{trans} - \text{sample}(\alpha_3 \delta_{trans}^2 + \alpha_4 \delta_{rot1}^2 + \alpha_4 \delta_{rot2}^2)$ 
7  $\hat{\delta}_{rot2} = \delta_{rot2} - \text{sample}(\alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2)$ 
8  $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$ 
9  $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$ 
10  $x' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$ 
11 return  $x_t = (x', y', \theta)^T$ 

```

TABLE II: Die Algorithmik für die Berechnung des Status x_t über die Daten der Odometrie, wobei α_1 bis α_4 Roboter spezifische Parameter für die Fehlerabbildung in der Kinematik sind. Quelle: [8]

```

1 sample ( $b^2$ )
2 return  $\frac{1}{2} \sum_{i=1}^{12} \text{rand}(-b, b)$ 

```

TABLE III: Berechnung der Wahrscheinlichkeit (Gauss-Verteilung) durch den Algorithmus **sample** , Quelle: [8]

Winkelauflösung. Aus diesem Grund werden Algorithmen zur Berechnung der mathematischen Modelle von Laserscannern dargestellt. für die Erklärung der Algorithmen in Table IV (S. 9) und V (S. 10) sowie deren mathematischen Herleitungen ist in [8] auf den Seiten 151 - 184 nachzuschlagen.

Der Algorithmus beam_range_finder_model weist jedoch einige Beschränkungen auf. In vollen Räumen, z. B. mit vielen Tischen und Stühlen, kommt es bei der Abtastung dieser Umgebung zu starken Schwankungen, da oftmals nur die Beine der Objekte erkannt werden (Siehe Fig. 5, S. 9). Verändert der Roboter seine Pose x_t nur gering, führt dies zu einer enormen Änderung der gemessenen Distanzen. Diese entstehenden schnellen Änderungen der gesamten Messung beeinflussen die Schätzung und kann so zu stärkeren Fehlern in der Schätzung führen. Aufgrund der hohen Anzahl von lokalen Maxima in der Distanzmessung führt dies zu der Ausschließung von gewissen Algorithmen, wie z. B. „Hill-Climbing“. Zudem ist das Berechnung von „ray casting“ für jeden Laserstrahl sehr rechenaufwändig. Um diesen Einschränkungen entgegenzuwirken, wurde likelihood_field_range_finder_model (Table V, S.10) entwickelt.

```

1 beam_range_finder_model ( $z_t, x_t, m$ )
2  $q = 1$ 
3 for  $k=1$  to  $K$  do
4   berechne  $z_t^{k*}$  für die Messung  $z_t^k$  durch „ray casting“
5    $p = z_{hit}p_{hit}(z_t^k|x_t, m) + z_{short}p_{short}(z_t^k|x_t, m) + z_{max} * p_{max}(z_t^k|x_t, m) + z_{rand} * p_{rand}(z_t^k|x_t, m)$ 
6    $q = q * p$ 
7 end
8 return  $q$ 

```

TABLE IV: Berechnung der Wahrscheinlichkeit eines Sets von Distanzmessungen zum Zeitpunkt t durch den Algorithmus `beam_range_finder_model`, Quelle: [8]

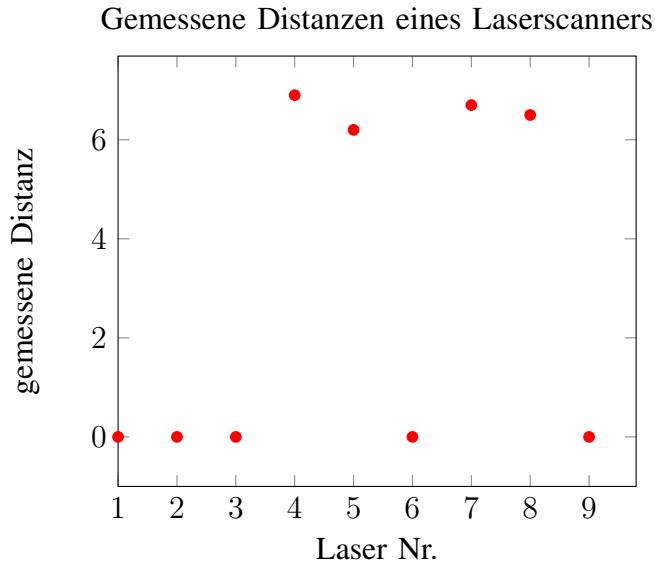


Fig. 5: Ein Set von gemessenen Distanzen eines Laserscanners. Die Laser Nr. 4, 5, 7, 8 haben kleine Objekte entdeckt, z. B. Stuhlbeine. Die gemessene Distanz 0 bedeutet, dass die maximale messbare Reichweite überschritten, also nichts gefunden wurde

C. Lokalisierung

Für die Lokalisierung des autonomen mobilen Systems in der Umgebungskarte wird der Knoten **AMCL** in ROS verwendet [9]. Ziel ist es hierbei, das Ursprungsframe des Roboterkoordinatensystems (`base_frame`) über die Odometriedaten (`odom_frame`) auf das Frame der Umgebungskarte bzw. Weltkoordinatensystem(`map_frame`) zu transformieren. Da die Informationen der Odometrie fehlerbehaftet sind (Dead Reckoning = Kumulativer Fehler der Odometrie im Laufe der Zeit, führt zu

```

1 likelihood_field_range_finder_model ( $t_t, x_t, m$ )
2  $q = 1$ 
3 for  $k=1$  to  $K$  do
4    $x_{z_t^k} = x + x_{k,sens} \cos \theta - y_{k,sens} + z_t^k \cos(\theta + \theta_{k,sens})$ 
5    $y_{z_t^k} = y + y_{k,sens} \cos \theta - x_{k,sens} + z_t^k \cos(\theta + \theta_{k,sens})$ 
6    $dist = \min_{x',y'} \{ \sqrt{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2} | \langle x', y' \rangle \text{ occupied in } m \}$ 
7    $q = q \cdot (z_{hit} \cdot prob(dist, \sigma_{hit}) + \frac{z_{random}}{z_{max}})$ 
8 end
9 return  $q$ 

```

TABLE V: Berechnung der Wahrscheinlichkeit eines Sets von Distanzmessungen zum Zeitpunkt t durch den Algorithmus **likelihood_field_range_finder_model**. $prob(dist, \sigma_{hit})$ berechnet die Wahrscheinlichkeit der Distanz unter Verwendung einer Gauss-Verteilung mit $\mu = 0, \sigma_{hit}$. Quelle: [8]

einem Abdriften der Odometrie (Odometry Drift)), die Lokalisierung zusätzlich durch eine Schätzung der Roboterpose in Bezug auf die Pose der Umgebungskarte unterstützt (siehe Fig. 6 auf Seite 10). Diese Unterstützung basiert auf der „adaptive Monte Carlo Lokalisierung“ Methode.

AMCL Map Localization

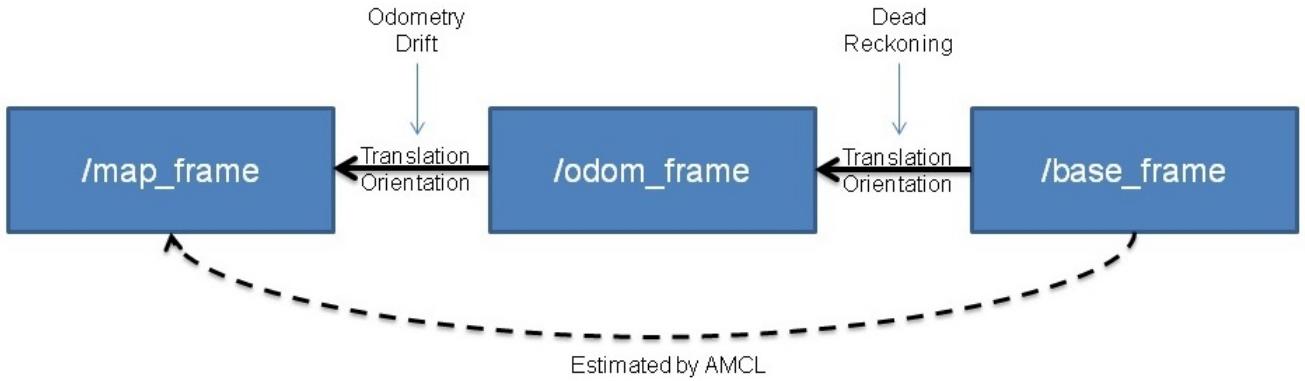


Fig. 6: Verarbeitungskette der Odometrie in Kombination mit AMCL, $base_frame$ = Frame in Roboterkoordinatensystem, $odom_frame$ = durch die Odometrie bestimmte Pose im Weltkoordinatensystem, map_frame = Weltkoordinatensystem, dem die Umgebungskarte zugrunde liegt. Quelle: [9]

1) Monte Carlo Lokalisierung:

Der „Monte Carlo Lokalisierung“ Algorithmus basiert auf der Schätzung durch Partikelfilter. In Table VI, S.11 wird die Algorithmik eines klassischen Partikelfilters vorgestellt. Ausgehend von diesem Algorithmus haben sich viele Erweiterungen um den Partikelfilter entwickelt, die Ähnlichkeiten sind in Table VII, S. 12 oder auch in der Erläuterung der Ablaufschritte in Kapitel II-F, S. 14 erkennbar.

```

1 Partikelfilter ( $\chi_{t-1}, u_t, z_t, m$ )
2  $\chi_t = \bar{\chi}_t = \emptyset$ 
3 for  $m=1$  to  $M$  do
4   sample  $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$ 
5    $w_t^{[m]} = p(z_t|x_t^{[m]})$ 
6    $\bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
7 end
8 for  $m=1$  to  $M$  do
9   draw  $i$  with probability  $\propto w_t^{[m]}$ 
10  add  $x_t^{[m]}$  to  $\chi_t$ 
11 end
12 return  $\chi_t$ 
```

TABLE VI: Der klassische Partikelfilter als Pseudo-Code, Quelle: [8]

Im Folgenden wird auf die Funktionsweise des Codes in Table VI eingegangen: In Zeile 4 wird durch die Schleifeniteration eine Menge $\bar{\chi}_t$ an Partikeln $x_t^{[m]}$ im aktuellen Zeitpunkt t der Größe M berechnet. Die Grundlage der Berechnung ist die Wahrscheinlichkeit vom Status x_t , unter der Bedingung der aktuellen Steuerungsdaten u_t und des Partikels $x_{t-1}^{[m]}$ aus der Partikelmenge χ_{t-1} . Diesen Schritt bezeichnet man als „Sampling“. Die in Zeile 5 als „Importance Weighting“ bezeichnete Berechnung kalkuliert den sogenannten Gewichtungsfaktor $w_t^{[m]}$. Dieser Schritt involviert die Umgebungsdaten z_t und berechnet die Wahrscheinlichkeit der Messung z_t bei gegebenen $x_t^{[m]}$. Vereinfacht wird $w_t^{[m]} = \frac{\text{Verteilung der Sensordaten}}{\text{Verteilung der Partikel}}$ berechnet. Die Zeilen 8 bis 11 werden unter dem Schritt „Resampling“ zusammengefasst. Es wird jedes Partikel in $\bar{\chi}_t$ neu erzeugt. Unter Einbindung der berechneten Gewichtung $w_t^{[m]}$ jedes Partikels wird die Menge der Partikel M in eine neue Menge transformiert. In diesem Prozess wird die Wahrscheinlichkeitsverteilung der Partikel verändert.

Die Erweiterung eines Partikelfilters zu der Monte Carlo Lokalisierung (Table VII, S. 12) basiert auf einer Anpassung einzelner Schritte. Der Schritt Sampling wird durch einen Algorithmus `sample_motion_model` durchgeführt. Im Regelfall werden hierbei die in Kapitel II-B1, S.6 vorgestellten Algorithmen verwendet. Die nächste Änderung basiert auf der Auswahl des `measurement_model` Algorithmus. Neben den Möglichkeiten aus Kapitel II-B2, S. 7 sind weitere Algorithmen einsetzbar. Diese sind unter anderem in [8, S.150 - 183] einsehbar.

```

1 Monte Carlo Lokalisierung ( $\chi_{t-1}, u_t, z_t, m$ )
2  $\chi_t = \bar{\chi}_t = \emptyset$ 
3 for  $m=1$  to  $M$  do
4    $x_t^{[m]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$ 
5    $w_t^{[m]} = \text{measurement\_model}(z_t, x_t^{[m]}, m)$ 
6    $\bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
7 end
8 for  $m=1$  to  $M$  do
9   draw  $i$  with probability  $\propto w_t^{[m]}$ 
10  add  $x_t^{[m]}$  to  $\chi_t$ 
11 end
12 return  $\chi_t$ 

```

TABLE VII: Der Monte Carlo Algorithmus als Pseudo-Code, Quelle: [8]

2) Adaptive Monte Carlo Lokalisierung:

Um die auftretende Rechenlast in autonomen Systemen gering zu halten, ist es sinnvoll die benötigte Partikelanzahl des Partikelfilters anzupassen. Zu Beginn der Lokalisierung werden sehr viele Partikel benötigt, oft mehr als 100000. Ist die ungefähre Position des Roboters jedoch bekannt, kann die Anzahl reduziert werden. Dies geschieht mit der Methode des KDL-Samplings bzw. Adaptive Monte Carlo Lokalisierung. Hierbei werden so lange Partikel erzeugt, bis ein gewisser statistischer Schwellwert M_χ erreicht wird. Somit kann die Anzahl der Partikel reduziert werden. Für eine genaue Beschreibung wird auf [8, S. 263 - 267] verwiesen.

D. Pfadplanung

Wie bereits in Kapitel I-C, S. 2 beschrieben wurde, benötigt ein mobiles autonomes System zwei Arten von Wegplanungen: Einen lokalen Planer, um auf Objekte in der unmittelbaren, mit den Sensoren erkennbaren Umgebung, zu reagieren. Und einen globalen Planer, um eine Trajektorie zwischen Start- und Zielpunkt auch über große Distanzen zu berechnen. Diese Planer befinden sich im Modul `nav_core`, Fig. 7, S. 13.

Der in diesem Projekt verwendete Planer basiert auf dem „dynamic windows approach“. Grundlage hierfür ist eine Costmap. Es wird zwischener einer globalen und einer lokalen Costmap unterschieden. Die durch einen SLAM-Algorithmus erstellte oder noch in der Erstellungsphase befindende Umgebungskarte wird zur Generierung der globalen Costmap verwendet. Die Local-Costmap wird durch die erhaltenen Sensordaten der Umgebung erzeugt. Sie ist inhaltlich genau wie die Global-Costmap aufgebaut, nur dass das durch Sensoren erreichbare Umfeld abgebildet wird. Costmaps sind in einzelne Zellen unterteilt, die in ROS folgende Werte besitzen:

- in Zelle: Status = Besetzt \rightarrow Wert = 100

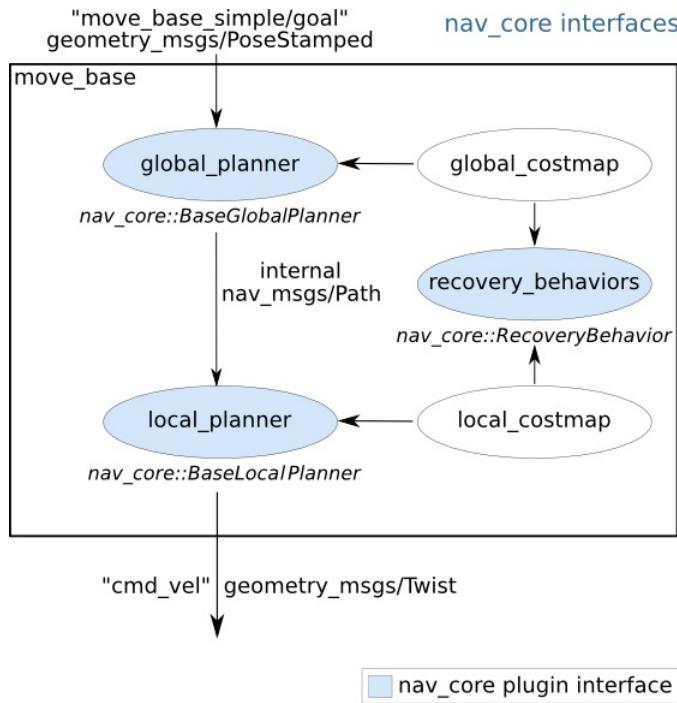


Fig. 7: Das in ROS verwendete Navigationsmodul `move_base` sowie dessen Programmstruktur. Als Input dienen die Zielkoordinaten und die gewünschte Pose des Roboters im Ziel. Nach Verarbeitung im `global_planner` wird die erstellte Trajektorie an den `local_planner` weitergegeben. Dieser prüft unter Nutzung der `local_costmap` die lokale Trajektorie und gibt die kurzfristig anstehende Translation und Rotation weiter. Blau markierte Module sind Bestandteil des `nav_core` Moduls. Quelle: [10]

- in Zelle: Status = zwischen Besetzt und Frei (z. B. unwegsames Gelände) $\rightarrow 0 < \text{Wert} < 100$
- in Zelle: Status = Frei $\rightarrow \text{Wert} = 0$
- in Zelle: Status = Unbekannt $\rightarrow \text{Wert} = -1$

Der Algorithmus „dynamic windows approach“ erstellt, abhängig von der aktuellen Pose des Roboters, Variationen der einzelnen möglichen Trajektorien. Da diese Trajektorien durch die Zellen der Costmap verlaufen, kann mittels einer Kostenfunktion jede mögliche Trajektorie gewichtet werden. Nach Gewichtung der einzeln geplanten Bahnen kann so eine geeignete ausgewählt werden. Der Unterschied zwischen den Planern besteht primär in der Planungsreichweite. Da der globale Planer die globale Costmap verwendet, kann dieser auch über eine größere Distanz planen. Trotz des Wissens über die gesamte Umwelt besitzt der globale Planer nur eine begrenzte Reichweite, da sonst die Berechnungen der gewichteten Trajektorien viel Zeit in Anspruch nehmen würde. Daher kann es bei langen Strecken vorkommen, dass der globale Planer die Strecke nach planen muss. Der lokale Planer verwendet im Gegensatz nur die lokale Costmap, daher ist die Reichweite sehr begrenzt. Da dieser aber durch

die Sensorinformationen, die ein aktuelles Abbild der Umwelt darstellen, kann er auf Änderungen reagieren. Der lokale Planer besitzt eine höhere Wiederholungsfrequenz als die globale Planung, d.h. die Berechnung wird kontinuierlich wiederholt.

E. Kombination Lokalisierung und Pfadplanung

Da eine gute Pfadplanung eine möglichst genaue Positionsschätzung des autonomen Systems voraussetzt, werden die Informationen der Lokalisierungsschätzung durch das amcl Modul an das move_base Modul (Fig. 7, S. 13) übergeben. Zusätzlich wird für den globalen Planer die Umgebungskarte durch das Modul map_server geladen und in die globale Costmap (global_costmap) transformiert. Die Bereitstellung der Sensorinformationen- und transformationen sowie die Odometrie für das move_base Modul sind ebenfalls wichtiger Bestandteil der Pfadplanung. In Figure 8 auf Seite 14 ist das Zusammenspiel der einzelnen Modulkomponenten aufgezeigt.

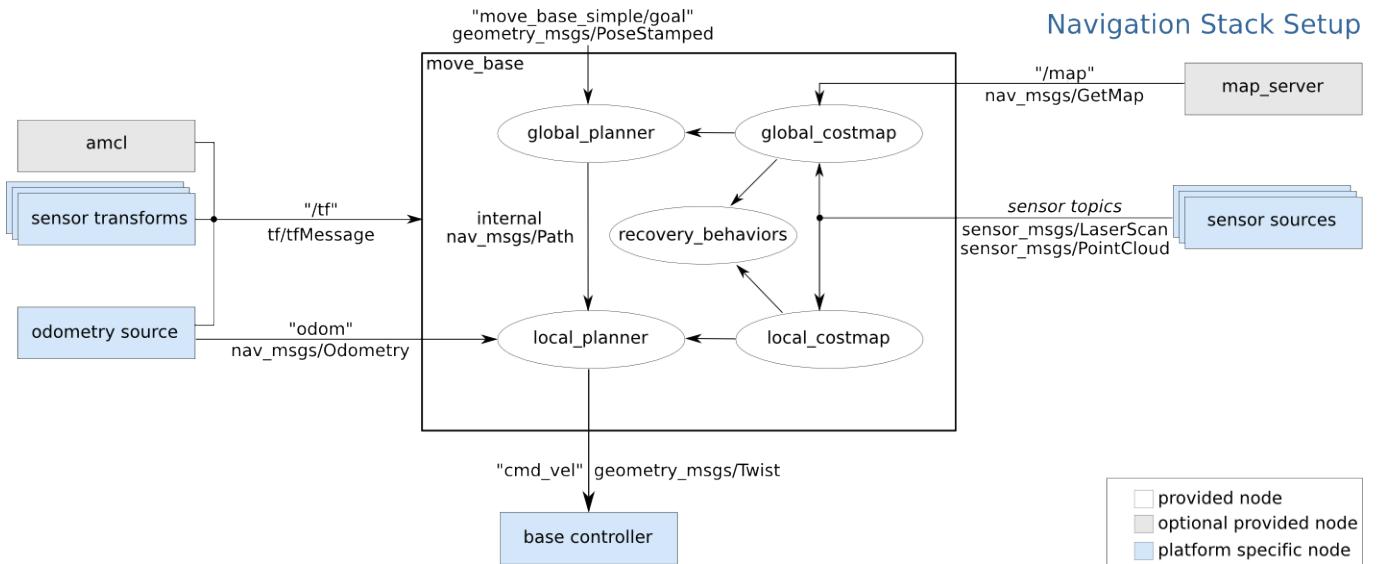


Fig. 8: Zusammenhänge zwischen der Pfadplanung, der Lokalisierungsschätzung über amcl, Sensorinformationen und der Umgebungskarte. Quelle: [11]

F. SLAM

Mittlerweile existieren eine Vielzahl von Vefahren für die Kartierung der Umgebung und gleichzeitiger Lokalisierung in dieser. In [2] werden aktuell genutzte SLAM-Algorithmen in ROS in Bezug auf Genauigkeit und Rechenlast analysiert und bewertet. **GMapping** wurde als robust beschrieben, da die Ungenauigkeit in beiden Szenarien, Simulation und reales Umfeld, im Vergleich zu anderen SLAM-Algorithmen konstant niedrig blieb sowie die CPU-Auslastung nicht nennenswert hoch war,

siehe Table VIII, S. 15. Aus diesen Gründen wurde GMapping zur Kartenerstellung in diesem Projekt verwendet. GMapping basiert auf dem Rao-Blackwellized Mapping [12].

TABLE VIII: Die Testergebnisse der in ROS verwendeten SLAM-Algorithmen im Vergleich, Werte der Simulation sowie reales Umfeld I-III: Schätzfehler. Werte CPU-Last: Auslastung in %. Quelle [2]

	HectorSLAM	GMapping	KartoSLAM	CoreSLAM	LagoSLAM
Simulation	7.4581	5.3670	5.4380	171.5218	9.3041
reales Umfeld I	1.1972	2.1716	1.0318	4.75333	3.0264
reales Umfeld II	0.5094	0.6945	0.3742	7.9463	0.8181
reales Umfeld III	1.0656	1.6354	0.9080	7.5824	2.5236
CPU-Last: Median	6.1107	7.0873	5.4077	5.5213	21.0839

1) Rao-Blackwellized Sample Mapping:

Die zentrale Idee beim Rao-Blackwellized Partikelfilter für SLAM ist, bei gegebenen Sensordaten ($z_{1:t}$ = Daten der Umgebung, z. B. Laserscan; $u_{0:t}$ = Informationen der Odometrie), eine Schätzung $p(x_{1:t}|z_{1:t}, u_{0:t})$ möglicher Trajektorien $x_{1:t}$ des Roboters zu berechnen. Diese Schätzung wird verwendet, um eine Gesamtwahrscheinlichkeit über die Karten m und Trajektorien zu berechnen:

$$p(x_{1:t}, m|z_{1:t}, u_{0:t}) = p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{0:t}) \quad (1)$$

Um die Wahrscheinlichkeit $p(m|x_{1:t}, z_{1:t})$ zu berechnen, nutzt das Rao-Blackwellized Mapping den Rao-Blackwellized Sample Importance Resampling (SIR) Partikelfilter, in dem zu jedem Satz an Partikeln eine individuelle Karte erstellt wird. Diese wird anhand der Beobachtungen $z_{1:t}$ und der Trajektorie $x_{1:t}$, repräsentiert durch die Partikel, generiert. Die Funktionsweise des Filters kann in vier Schritten beschrieben werden. Sie ähneln die der Monte Carlo Lokalisierung (S. 12), da beide auf dem klassischen Verfahren des Partikelfilters basieren.

- 1) *Sampling*: Es wird ein neues Set an Partikeln erzeugt $\{x_t^{(i)}\}; i = 1, \dots, N$, auf Basis der vorherigen Partikel $\{x_{t-1}^{(i)}\}$ und der Verteilungsfunktion $\pi(x_t|z_{1:t}, u_{0:t})$.
- 2) *Importance Factor*: Eine individuelle Gewichtung wird für jedes Partikel berechnet:

$$w^{(i)} = \frac{p(x_t^{(i)}|z_{1:t}, u_{0:t})}{p_i(x_t|z_{1:t}, u_{0:t})} \quad (2)$$

- 3) *Resample*: Partikel mit geringer Gewichtung werden durch Partikel mit hoher Gewichtung ersetzt.
- 4) *Map Estimating*: Für jedes $\{x_t^{(i)}\}$ wird die Wahrscheinlichkeit $p(m_t^{(i)}|x_t^{(i)}, z_{1:t})$, basierend auf der Trajektorie und der vergangenen Beobachtungen der Umwelt, berechnet.

2) GMapping:

Die Erweiterung auf das GMmapping entsteht durch eine adaptive Technik, um die Partikelmengen zu reduzieren. Erstens wird die Wahrscheinlichkeitsverteilung verbessert. Die optimale Verteilung ist laut [12] durch

$$p(x_t|m_{t-1}^{(i)}, x_{t-1}^{(i)}, z_t, u_t) = \frac{p(z_t|m_{t-1}^{(i)}, x_t)p(x_t|x_{t-1}^{(i)}, u_t)}{\int p(z_t|m_{t-1}^{(i)}, x')p(x'|x_{t-1}^{(i)}, u_t)dx'} \quad (3)$$

bestimmt. Aufgrund der hohen Genauigkeit von Laserscannern dominiert $p(z_t|m_{t-1}^{(i)}, x_t)$ das Produkt, was bedeutet, dass $p(x_t|x_{t-1}^{(i)}, u_t)$ im Intervall $L^{(i)}$ als Konstante k betrachtet werden kann. Dabei ist $L^{(i)}$ definiert als:

$$L^{(i)} = \{x | p(z_t|m_{t-1}^{(i)}, x) > \varepsilon\} \quad (4)$$

Daraus ergibt sich folgende Annäherung:

$$p(x_t|m_{t-1}^{(i)}, x_{t-1}^{(i)}, z_t, u_t) \approx \frac{p(z_t|m_{t-1}^{(i)}, x_t)}{\int_{x' \in L^{(i)}} p(z_t|m_{t-1}^{(i)}, x')dx'} \quad (5)$$

Zusätzlich wird um das Maxima der Verteilungsfunktion die Funktion durch eine Gauss-Verteilung approximiert.

$$p(x_t|m_{t-1}^{(i)}, x_{t-1}^{(i)}, z_t, u_t) \approx \mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)}) \quad (6)$$

mit

$$\mu_t^{(i)} = \frac{1}{\eta} \sum_{j=1}^K x_j p(z_t|m_{t-1}^{(i)}, x_j) \quad (7)$$

$$\Sigma_t^{(i)} = \frac{1}{\eta} \sum_{j=1}^K p(z_t|m_{t-1}^{(i)}, x_j) (x_j - \mu_t^{(i)}) (x_j - \mu_t^{(i)})^T \quad (8)$$

η wird in diesem Fall zur Normierung genutzt

$$\eta = \sum_{j=1}^K p(z_t|m_{t-1}^{(i)}, x_j) \quad (9)$$

Unter der Verwendung dieser Annahmen und der daraus resultierenden Verteilungsfunktion ist die Gewichtungsfunktion für $w_t^{(i)}$ definiert als:

$$w_t^{(i)} = w_{t-1}^{(i)} k \eta \quad (10)$$

Zweites wird eine Bedingung an den Schritt *Sampling* auf S. 15 angefügt. Dieser Schritt wird nur ausgeführt, wenn $N_{eff} < N$, wobei N die Partikelanzahl ist und sich N_{eff} berechnet aus:

$$N_{eff} = \frac{1}{\sum_{i=1}^N (x^{(i)})^2} \quad (11)$$

Durch diese Erweiterungen ist GMapping trotz niedriger Rechenzeiten ein effizienter SLAM-Algorithmus und kann auch auf Plattformen verwendet werden, die eine geringe Rechenkapazität aufweisen.

III. LÖSUNGSANSATZ DER PROBLEMSTELLUNG

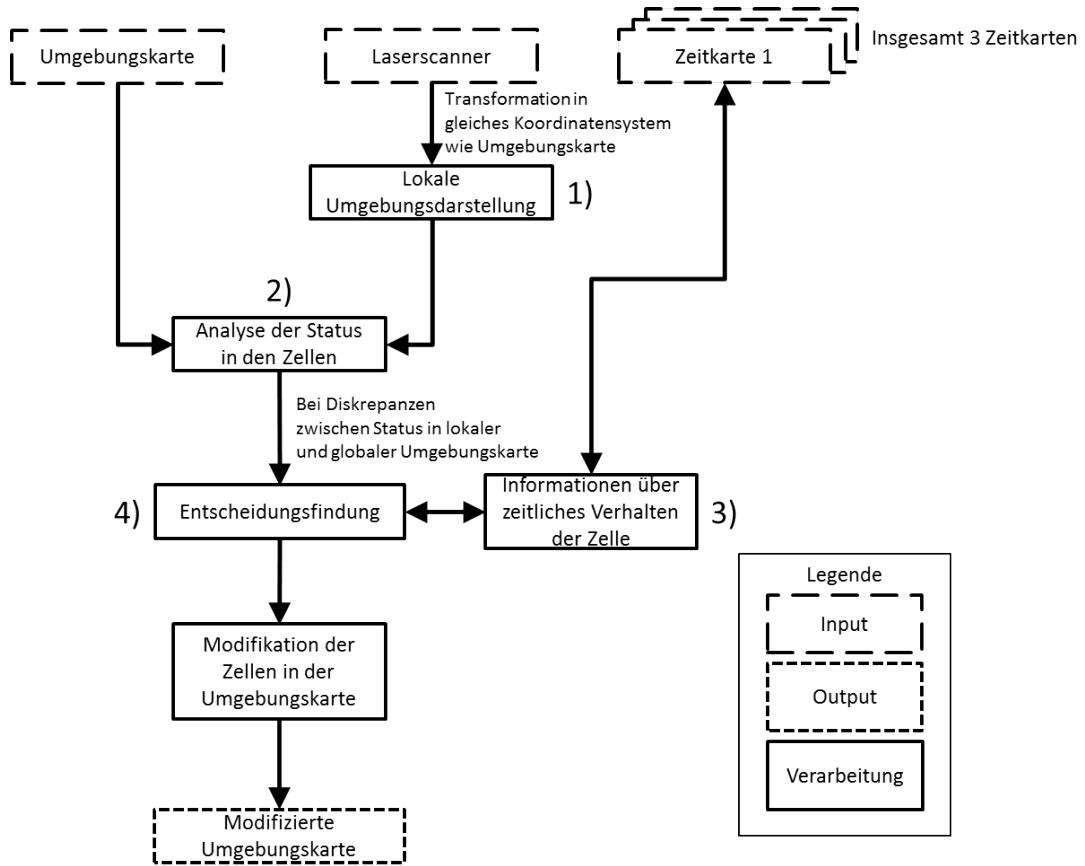


Fig. 9: Konzept der Verarbeitungskette. Zu Beginn werden die benötigten Informationen der Umgebungskarte und des Laserscanners gesammelt (Input). Anschließend wird jede Zelle des lokalen und globalen Abbilds auf Unterschiede in den Status analysiert. Finden sich diese, wird in Abhängigkeit der Informationen aus dem zeitlichen Verhalten der Zelle entschieden, ob eine Modifikation der Zelle der Umgebungskarte nötig ist (Output).

A. Voraussetzungen

In diesem Forschungsprojekt wird angenommen, dass eine Umgebungskarte bereits erstellt ist, diese vollständig ist und eine hohe Genauigkeit aufweist. Zudem ist die Initialisierungsposition des Roboters auf der Umgebungskarte, der Startpunkt, bekannt. Die Lokalisierungsgenauigkeit wird nicht als ideal

betrachtet, die entstehende Ungenauigkeit wird berücksichtigt. Ebenso werden die Fehler der Sensoren mit einbezogen.

B. Konzeption

1) Ausgangslage:

Wie in der Forschungsfrage beschrieben, geht es um die langfristige Wartung der Umgebungskarte im Bezug auf den Zeitverlauf. Ein möglicher Ansatz wäre die direkte Integration von Änderungen in die Umgebungskarte, d.h. die physische Spiegelung der Umgebung. Ein Vorteil für die Lokalisierung besteht in diesem Fall durch die adaptierte Umgebungskarte. Der wesentliche Nachteil ist jedoch, dass die Informationen der Umgebungskarte im Laufe der Zeit nur noch durch die Sensorinformationen repräsentiert wird, der Informationsanteil der Karte aus SLAM reduziert sich allmählich. Verdrängen die Sensorinformationen die SLAM-Informationen auf der Umgebungskarte, ist Stützung der Lokalisierung obsolet, da Abweichungen nicht mehr erkannt werden. Für die Pfadplanung ist dieses Szenario ebenfalls nicht geeignet, da unter Umständen kurzfristig blockierte Passagen eingetragen werden und nicht mehr als möglicher Pfad in Betracht gezogen werden, wie in Figure 10 auf Seite 19 aufgezeigt wird. In diesem Beispiel wird ein ursprünglich befahrbarer Pfad blockiert. Diese Blockierung wird in die Umgebungskarte integriert. Ist die Blockade aufgehoben, wird dies jedoch nicht mehr erkannt, da der Roboter den kritischen Bereich nicht erreicht. Diese Szenario kann auch im entgegengesetzten Fall eintreten, ein an sich blockiertes Gelände wird nur kurzzeitig befahrbar. Eine physische Adaption ist somit nicht geeignet. Es muss daher die Veränderung auf der Karte nur erkannt werden und diese dementsprechend als potentielle Kandidaten für eine Anpassung auf der Umgebungskarte markiert werden. Die endgültige Modifikation der Karte muss anhand einer Entscheidungsfindung realisiert werden, die, wie in der Forschungsfrage definiert, die verstrichene Zeit als Gewichtungsfaktor einbezieht.

2) Schnittstelle, Input/ Output:

Wie in Kapitel I-D, S.5 beschrieben, ist das Ziel die Adaption der ortsabhängigen Informationen durch die Verwendung der Zeit als zusätzliche Komponente. Eine sinnvolle Möglichkeit, die Adaption in den Prozess zu integrieren, ist die Veränderung der Zelleninformationen in der Umgebungskarte. Die Umgebungskarte wird doppelt verwendet: Einerseits dient sie als Eingangsinformation über den bekannten Zustand der Umwelt, andererseits wird sie im Laufe des Prozesses modifiziert und ist daher auch als das Ergebnis zu betrachten. Neben der Umgebungskarte werden zusätzlich die Daten der Laserdistanzmessung als Eingangsinformationen verwendet, um eine Abbildung der lokalen Umwelt zu erhalten. Als wichtiger Entscheidungsfaktor wird noch eine Karte mit Zeitstempeln übergeben. Sie hat die gleichen Ausmaße wie die Umgebungskarte (Länge und Breite), in den Zellen werden Zeitstempel gespeichert. Es existieren genau drei solcher Zeitkarte. Die Inputs und Outputs sind wie folgt definiert:

Input:

- Daten des Laserdistanzmessers
- Umgebungskarte

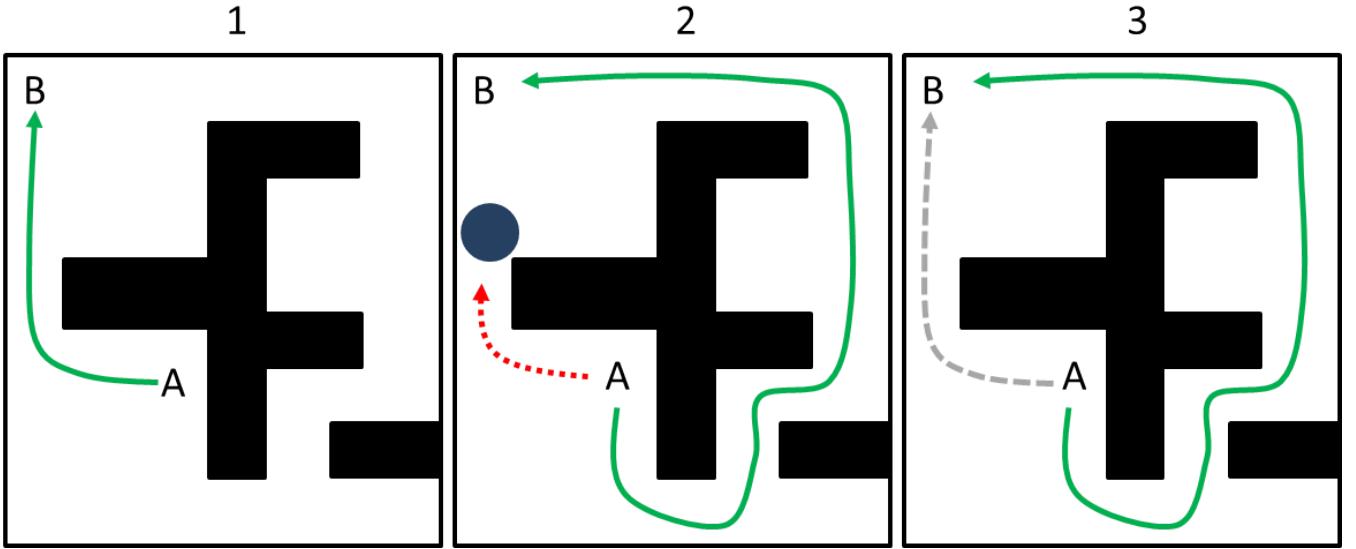


Fig. 10: Problem der Pfadplanung. Aufgabe ist es, ausgehend von A, Punkt B zu erreichen. Ein Pfad, der in 1 befahrbar (grüner Pfeil beschreibt die Trajektorie) war, wird in 2 blockiert. Der Roboter registriert die Blockierung (blauer Kreis) und trägt diese sofort in die Umgebungskarte ein. Da dieser Pfad blockiert ist (rote Linie), muss ein Umweg gefahren werden. In 3 ist der Pfad aus 1 wieder befahrbar (graue Linie), der Roboter erwägt jedoch noch eine existierende Blockade. Er plant eine überflüssige Trajektorie.

- Karten mit Zeitstempeln

Output:

- geänderte Umgebungskarte

3) Lokale Umgebungsdarstellung: Fig. 9: 1)

Nachdem Input und Output definiert sind, wird in Fig. 9, Seite 17 das Konzept aufgezeigt. Die Daten der Umgebungskarte liegen als kartesische Koordinaten vor. Da die Daten des Laserscanners nicht auf dem Koordinatensystem der Umgebungskarte basiert, sondern eine eigene Darstellungsform besitzen [13], müssen diese noch in kartesische Koordinaten mit gleichem Ursprung wie die Umgebungskarte transformiert werden. Daraus entsteht die lokale Umgebungsdarstellung. Bedingt durch die Funktionsweise der Sensoren sind nur die Koordinaten besetzter Zellen bekannt, daher enthält die Karte zunächst nur Zellen mit dem Status Besetzt. Die Koordinaten dieser Zellen werden im Vektor $p_{\text{scan}}(i) = (x_i \ y_i)^T$ abgespeichert, mit $i = \text{Anzahl der Sensordaten eines Scandurchlaufs}$. Die Zellen mit Status Frei werden durch eine lineare Interpolation zwischen Roboterposition und den Zellen Besetzt berechnet. Zellen, die weder besetzt noch frei sind, werden mit Unbekannt markiert (Fig: 12, S. 21). Da die Informationen des Laserscanners mit einem gewissen Fehlerrauschen behaftet sind und daher eine Ungenauigkeit aufweisen, ist es sinnvoll die Informationen zu filtern. Dies geschieht,

indem die erhaltenen und transformierten Daten zu verschiedenen Zeitpunkten aufsummiert werden. Es liegen zu jedem Zeitpunkt Datensätze vor, die durch das Rauschen eine leichte Abweichung besitzen. Es wird eine Karte M_{filter} erstellt, die in allen Zellen den Wert 0 besitzen. Bei jeder Iteration werden die Werte in den Zellen erhöht, so dass bei identischen Koordinaten eine Kumulation entsteht, $M_{filter}(x, y) = M_{filter}(x, y) + \text{Gewichtung}$ (Table: 11, S. 20). Nach den Iterationsschritten werden die Koordinaten der Zellen in $p_{filter}^*(i) = (x_i \ y_i)^T$ gespeichert, die den Schwellwert erreichen oder überschreiten. Der dazugehörige Algorithmus ist in Table: IX auf Seite 21 beschrieben.

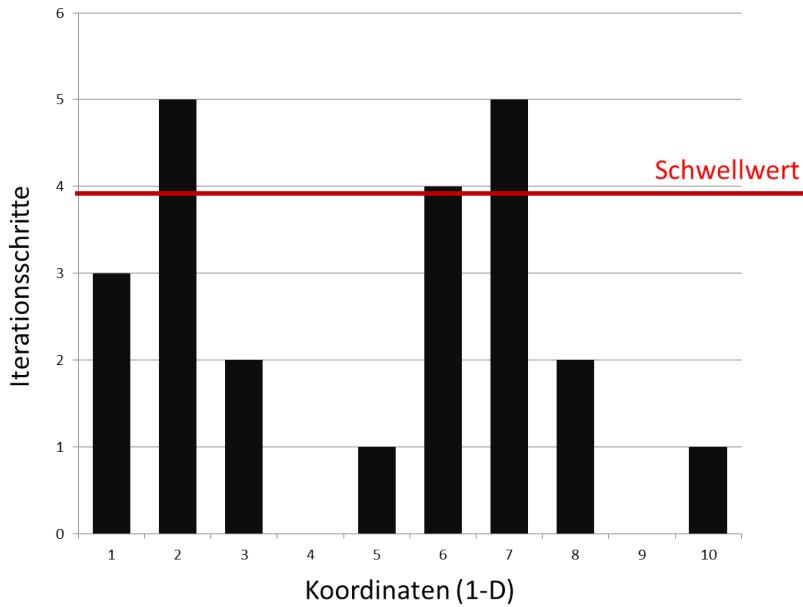


Fig. 11: Filterung der Sensorsignale. In jedem Iterationsschritt werden die Werte an den vorhandenen Koordinaten mit einem Gewichtungsfaktor aufsummiert. Sind die eingestellten Iterationsschritte durchlaufen, werden die Koordinaten als besetzte Zellen registriert, die überhalb des Schwellwerts liegen. Koordinate 10 ist nur einmal vorhanden, während Koordinaten 7 und 2 fünfmal in den Iterationsschritten gezählt wurden.

4) Analyse der Status in den Zellen: Fig. 9: 2)

Die beiden Karten,

globale Umgebungskarte := M_{global} ,

lokale Umgebungsdarstellung := M_{lokal}

können in den jeweiligen Zellen mit den Koordinaten x, y verschiedene Status besitzen. Diese sind:

Besetzt : Zelle ist als nicht befahrbar markiert.

Frei : Zelle kann befahren werden

Unbekannt : Keine Informationen über die Zelle verfügbar

Es sind folglich zwei Karten vorhanden, die beide die gleiche Abmessung haben. Da die Lokalisierung als präzise angenommen wird, beschreiben die Koordinaten die gleiche Zelle in den jeweiligen

```

1 Sensor Filterung ( $weight, iterationSteps, threshold$ )
2 for all cells:  $M_{filter} = 0$ 
3  $p_{filter} = 0$ 
4 for  $i=1$  to  $iterationSteps$  do
5    $p_{scan}$  = new incoming sensor data
6   for all elements in  $p_{scan}$  do
7     |  $M_{filter}(x, y) = M_{filter}(x, y) + weight$ 
8   end
9 end
10 for every cell in  $M_{filter}$  with  $M_{filter}(x, y) \geq threshold$  do
11   | add new item to  $p_{filter}$ 
12 end
13 return( $p_{filter}$ )

```

TABLE IX: Algorithmus zur Filterung der Sensorsignale. Zu jedem Iterationszeitpunkt wird ein anderer Datensatz von Sensoren erwartet, d.h. es existieren Wartezeiten zwischen den Iterationsschritten.

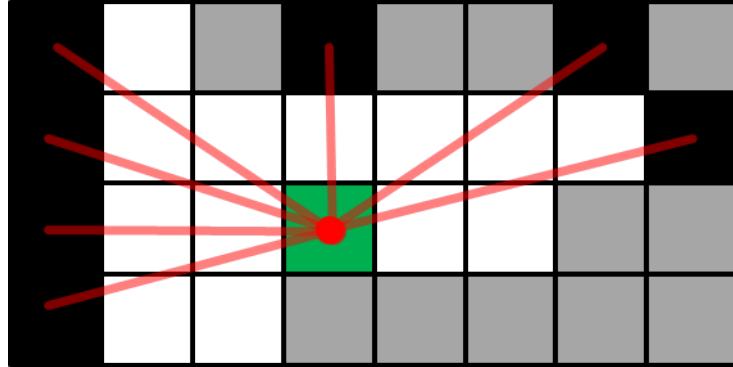


Fig. 12: Erstellung der lokalen Umgebungskarte. Grün = Position des Roboters. Rot = Interpolationslinie zwischen Roboter und besetzter Zelle. Schwarz = Von Sensor gefundenes Objekt, Zelle = Besetzt. Weiß = Frei, Status wird gesetzt, wenn die Zelle auf der Interpolationslinie liegt. Grau = Zelle mit Status Unbekannt, da keine Informationen enthalten sind.

Karten. Da die Dynamik der Umwelt in die Umgebungskarte eingebunden wird, ist es sinnvoll, die Zellen der zwei Karten miteinander zu vergleichen. M_{lokal} beschreibt den aktuellen Zustand der Umwelt, aber nur mit begrenzter Reichweite. M_{global} hingegen beschreibt den alten Zustand, aber in der gesamten Umwelt. Um eine Änderung der Umwelt zu registrieren, ist die Betrachtung der Status der Zellen der Schlüssel. Ist in der M_{lokal} eine Zelle besetzt, diese aber in M_{global} frei, so ist die Wahrscheinlichkeit hoch, dass sich der Zustand der Zelle in der Realität geändert hat. Dies gilt auch in der entgegengesetzten Richtung. Daher sind die Zellen zu betrachten, bei denen $M_{global}(x, y) \neq M_{lokal}(x, y)$ gilt. In Table X auf Seite 22 sind die verschiedenen Kombinationen der auftretenden Status aufgezeigt. Die mit Y markierte Kombination ist von Interesse. Bei genauerer Beleuchtung stellt man fest, dass es nur eine bestimmte Anzahl von Kombinationen gibt, die in Betracht

gezogen werden müssen. Da von einer vollständigen Umgebungskarte ausgegangen wird und somit unbekannte Bereiche nicht auftreten, reduzieren sich die zu betrachtenden Kombinationen auf:

$$\begin{aligned} M_{global}(x, y) &= \text{besetzt}, M_{local}(x, y) = \text{frei} \\ M_{global}(x, y) &= \text{frei}, M_{local}(x, y) = \text{besetzt} \end{aligned}$$

TABLE X: Die möglichen Status der Umgebungskarte bzw. der lokalen Karte für die einzelnen Zellen. X bedeutet eine Diskrepanz zwischen den Status.

		Umgebungskarte (global)		
		Besetzt	Frei	Unbekannt
Umweltabbild (lokal)	Besetzt		Y	Y
	Frei	Y		Y
	Unbekannt			

Da beide Kombinationen von einer besetzten Zelle ausgehen und die Anzahl der Zellen mit diesem Status in Normalfall wesentlich kleiner ist als die der freien Zellen, bietet es sich an, die Koordinaten sämtlicher besetzter Zellen zu sammeln:

$$N_{besetzt} = \text{Anzahl besetzter Zellen in } M_{lokal} + \text{Anzahl besetzter Zellen in } M_{global} \quad (12)$$

$$z_{combined} = \{x, y | M_{lokal}(x, y) = \text{Besetzt}\} \cup \{x, y | M_{global}(x, y) = \text{Besetzt}\} \quad (13)$$

$$\text{mit } z_{combined}(i) = \begin{pmatrix} x_i \\ y_i \\ origin_i \\ currentState_i \end{pmatrix}, i = 1, 2, \dots, N_{besetzt} .$$

\vec{z} enthält alle Koordinaten x, y von Zellen in M_{global} und M_{lokal} mit Inhalt Besetzt sowie die Herkunft $origin$. Die Herkunft beschreibt, auf welcher der beiden Karten die besetzte Zelle gefunden wurde. Ein Problem ist die Ungenauigkeit der Lokalisierung, beschrieben durch

$$\tau_b^w = o^b - o^w \quad (14)$$

Der Vektor wird für die Modellierung des Lokalisierungsfehlers verwendet und stellt den Fehler zwischen echter Position und berechneter Position des Roboters in der Umgebungskarte dar. Da τ_b^w nicht bekannt sein kann, wird dieser geschätzt (Fig. 13, S. 23).

Um die Diskrepanzen der Status von Interesse zu finden, wird der Algorithmus in Table XI auf Seite 24 genutzt. Es werden die Status der beiden Zellen miteinander verglichen. Abhängig von der Herkunft der Zellkoordinaten wird die Zielkarte ausgesucht. In Table XII, Seite. 24 wird im Umkreis der Zellenkoordinaten auf der Zielkarte nach einem vorhandenen Status gesucht. Als oberste Priorität wird nach besetzten Zellen gesucht, gefolgt von dem Status Frei. Ist keiner der Status auffindbar, wird die Zelle als Unbekannt markiert. Abhängig des gefundenen Status im Umfeld der Zellkoordinaten wird in $z_{combined}$ eine Vermerk $currentState$ hinterlegt. Dadurch ist die Zuweisung der Zellkoordinate,

deren Ursprung und der aktuelle Status der Zelle auf der oppositionellen Karte vollständig abgedeckt. Diese Informationen sind die Basis für die darauf folgenden Verarbeitungsschritte. In Fig. 14, S. 23, ist ein Abbild nach der Analyse der Status gezeigt. Die hier markierten Zellen beschreiben nicht den gefundenen Status, sondern ob Abweichungen (rot markierte Zellen) oder Übereinstimmungen (grüne Zellen) zwischen den Status der Zellen in M_{global} und M_{lokal} vorliegt. Rot markierte Zellen werden an die Entscheidungsfindung (Fig. 9: 4) weitergegeben.

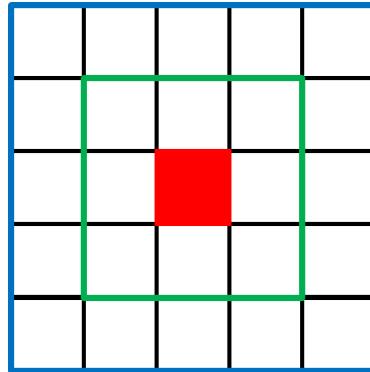


Fig. 13: Darstellung der Prüfung des Status der Zellen. Die zu untersuchende Zelle ist rot markiert. Durch die Quantisierung beschreibt τ_b^w die zu untersuchende Umgebung in Zellenabständen zur Ursprungszelle. Die grüne Umrandung zeigt die Untersuchung der Umgebung mit $\tau_b^w = 1$, blau $\tau_b^w = 2$.

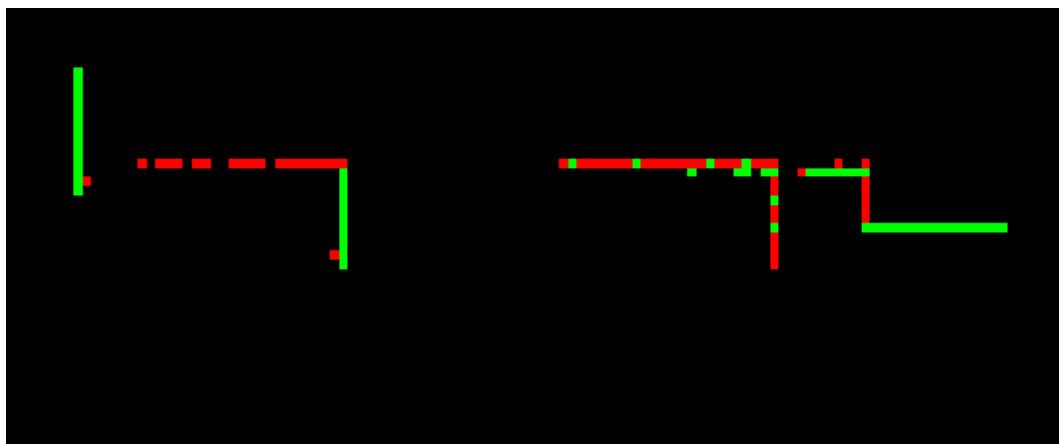


Fig. 14: Beispiel einer Statusanalyse. Es wurden alle Zellen mit Status Besetzt überprüft. Grün markierte Zellen bedeuten, dass eine Übereinstimmung zwischen den Status der Karten M_{global} , M_{lokal} existiert. Rote Felder zeigen eine Diskrepanz, $M_{global}(x, y) \neq M_{lokal}(x, y)$.

5) Informationen über zeitliches Verhalten der Zelle: Fig. 9: 3)

```

1 Analyse der Status in den Zellen ( $z_{combined}$ ,  $M_{global}(x, y)$ ,  $M_{local}(x, y)$ ,  $\tau_b^w$ )
2 for  $i=1$  to number of elements in  $z_{combined}$  do
3   global_Status= $M_{global}(\text{Coordinates } (x_i, y_i) \text{ from } z_{combined}(i))$ 
4   local_Status= $M_{local}(\text{Coordinates } (x_i, y_i) \text{ from } z_{combined}(i))$ 
5   if  $origin_i$  from  $z_{combined}(i) == M_{local}$  then
6     | checked_Status=checkState ( $M_{global}$ , global_Status,  $\tau_b^w$ )
7   else
8     | checked_Status=checkState ( $M_{local}$ , local_Status, ,  $\tau_b^w$ )
9   end
10  |  $z_{combined}(i) = \text{checked\_Status}$ 
11 end
12 return  $\chi_t$ 

```

TABLE XI: Algorithmus zur Analyse der Status in den Zellen. Abängig vom Ursprung ($z_{combined}(origin)$) wird über die oppositionelle Karte geprüft und in $z_{combined}(currentState)$ gespeichert.

```

1 checkState ( $M$ , Status_Coordinates,  $\tau_b^w$ )
2 if Status Occupied  $\in M(\text{Status\_Coordinates})$  within range  $\tau_b^w$  then
3   | status = Occupied
4 else if Status Free  $\in M(\text{Status\_Coordinates})$  within range  $\tau_b^w$  then
5   | status = Free
6 end
7 else
8   | status = Unknown
9 end
10 return status

```

TABLE XII: Algorithmus zur Erkennung des Status. Es wird in der Umgebung der Koordinaten Status_Coordinates in M mit der Distanz τ_b^w nach dem Status Besetzt gesucht und bei Erfolg zurückgegeben. Ist der Status nicht auffindbar, befindet sich der Status Frei in der Rückgabe.

Es existieren genau drei Zeitkarten, $M_{time\ occupied}$, $M_{time\ free}$, $M_{time\ unknown}$, jeweils eine für jeden Status (Besetzt, Frei, Unbekannt). Wie die Umgebungskarte und die lokale Umgebungsdarstellung besitzen diese Karten die gleichen Abmaße in Höhe und Breite. Der Unterschied besteht im Inhalt der Zellen. Jede Zelle kann einen Zeitwert abspeichern, sie beinhaltet einen Zeitstempel. Dieser Zeitstempel ist der Gewichtungsfaktor für die Entscheidungsfindung und die Grundlage des Einbezugs der Zeit in dynamischen Umgebungen. In Fig. 15 auf Seite 25 ist der Aufbau der Zeitkarten ersichtlich. Durch den gleichen geometrischen Aufbau ist eine Fusionierung der Status im Ortsbereich und des Zeitbereichs möglich, da jeder Koordinate in $M_{local}(x, y)$ bzw. $M_{global}(x, y)$ ein Zeitstempel zugewiesen wird. Bei der Initialisierung des Systems spiegeln die Zeitkarten M_{global} wider, in $M_{time\ occupied}$ besitzt jede Zelle einen Zeitstempel, dessen Zelle in M_{global} den Status Besetzt beinhaltet. Das Gleiche gilt für $M_{time\ free}$ und $M_{time\ unknown}$. Nicht zugewiesene Zellen der jeweiligen Zeitkarte besitzen den Wert 0. Das bedeutet, dass für jede Koordinate in den Zeitkarten nur eine Zelle mit Zeitstempel und zwei

Zellen mit dem Wert 0 existieren.

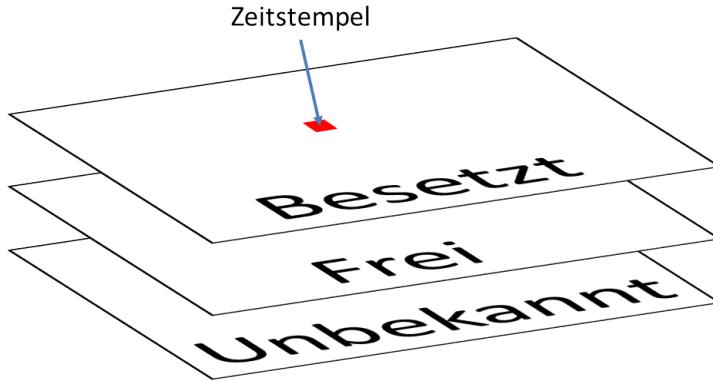


Fig. 15: Aufbau der Zeitkarten ($M_{time\ occupied}$, $M_{time\ free}$, $M_{time\ unknown}$) für die Status Besetzt, Frei, Unbekannt. Alle Zeitkarten besitzen die gleiche Geometrie und speichern Zeitstempel (Rot) in entsprechenden Zellen ab. Die Zellenkoordinaten sind identisch mit denen aus M_{lokal} bzw. M_{global} . Dadurch ist eine Zuweisung zwischen Ort und Zeit möglich. Ist die Zelle wie hier in $M_{time\ occupied}$ mit einem Zeitstempel versehen, besitzen die Zellen der gleichen Koordinate der anderen Zeitkarten den Wert 0.

6) Entscheidungsfindung: Fig. 9: 4)

Hier werden alle bisherigen Informationen von $\vec{z}_{combined}$, $M_{time\ occupied}$, $M_{time\ free}$, $M_{time\ unknown}$ für die Entscheidungsfindung, die Zellen in M_{global} zu verändern, verarbeitet. Die folgende Beschreibung bezieht sich auf den Algorithmus zur Entscheidungsfindung in Table XIII auf Seite 27. Da die in $\vec{z}_{combined}$ gespeicherten Zelleinträge, ursprünglich die Fusion der Zellen Besetzt aus M_{global} sowie M_{lokal} darstellen, sowie deren Status auf der oppositionellen Karte beinhalten, können unter Verwendung der zu betrachtenden Zustände aus X, S. 22 die nötigen Zustände extrahiert werden. Dieses sind die Einträge, dessen Wert *currenStatus* aus $\vec{z}_{combined}$ den Wert Frei besitzen (Zeile 2). Zu Beginn werden die Zeitstempel aus den Zeitkarten $M_{time\ occupied}$ und $M_{time\ free}$ für die jeweiligen Zellen geladen. Durch die in Zeile 9 und 12 durchgeführte Kontrolle wird eine Registrierung neu eintretender Zustände der Zellen durchgeführt. Dies bedeutet, dass bei Wert 0 die Zelle keine Änderung der Zelle in M_{global} erwarten. Wird ein neuer Zustand registriert, ändert sich der Inhalt der Zelle zu einem aktuellen Zeitstempel. Durch das sehr kurze Zeitintervall zwischen erhaltener Messung und Auswertung (im Bereich von ms bis s) im Vergleich zu der observierten Zeit bis zu einer definitiven Entscheidung der Statusänderung (Je nach Einsatzgebiet Stunden, Tage bis zu Wochen) kann diese vernachlässigt werden und es muss nicht auf den Zeitstempel der Messung zurückgegriffen werden. Diese daraus resultierende Toleranz gegenüber kleinen Zeiträumen verbessert die Robustheit des Ansatzes. Durch die Zellenregistrierung sind die Koordinaten als Kandidat für mögliche zukünftige Statusänderung in M_{global} markiert. Wurde einer der Zellen in $M_{time\ occupied}$ oder $M_{time\ free}$ bereits als Kandidat markiert, berechnet der Algorithmus in Zeile 16 und 17 die Differenz zwischen den in den Zeitkarten gespeicherten Zeitstempeln und dem aktuellen Zeitpunkt. Durch die vorherige Prüfung der Zellen auf den Wert 0 ist abgesichert, dass die Differenz nicht gleich dem aktuellen Zeitstempel ist. Der

Auswahlprozess, von welchem Status in welchen Status gewechselt wird, übernimmt Zeile 18 respektive 22. Die Logik besagt, dass bei einem Statuswechsel, der vorhandene Status länger existieren muss als der neu anzunehmende Status. Ist der alte Status z. B. Besetzt und es soll auf den neuen Status Frei gewechselt werden, so muss Besetzt bereits länger existiert haben. Ist das Ergebnis der Funktion g_{free} bzw. $g_{occupied}$ in den Zeilen 19 bzw. 23 mit den Parametern $dT_{occupied}, dT_{free}$ größer als der eingestellte Schwellwert $threshold_{occupied} / threshold_{free}$, so ist die Änderung des Status in M_{global} entschieden und der Status die Zelle mit den aus $\vec{z}_{combined}$ Koordinaten (x_i, y_i) wird geändert. Eine grafische Darstellung des Algorithmus anhand einer Zelle wird in Figure 16 auf Seite 28 gezeigt. Hierbei liegt der Fokus auf dem Verhalten und der Informationen über 13 Messzeitpunkte der Zeitkarten $M_{time\ occupied}, M_{time\ free}$ und der Karten M_{global}, M_{local} . Durch fehlerhafte Sensordaten kann der Fall eintreten, dass eine Zelle fälschlicherweise für einen Statuswechsel markiert wurde. Um diese Ausreißer zu eliminieren, setzt Zeile 4 bei Erkennung des alten Zustandes der Zelle den Inhalt der Zeitkarte $M_{time\ free}$ zurück auf 0. Somit ist die Zelle nicht mehr als Wechselkandidat markiert. Die Funktionen g_{free} und $g_{occupied}$ sind der Kernteil der Entscheidungsfindung. Es hat sich herausgestellt, dass die Entwicklung dieser Funktionen eine komplexe Herausforderung darstellt, auf die es keine triviale Lösung gibt.

Eine grundlegende Frage ist nach der Art der Umgebung. In sich schnell ändernden Umgebungen ist es sinnvoll, blockierte Bereiche erst nach längerer Existenz einzubinden. In sehr trügen Umgebungen lohnt es sich, Objekte schnell in die globale Umgebungskarte zu integrieren, da nicht von einer weiteren baldigen Zustandsänderung ausgegangen werden kann. Aber auch genau das Gegenteil ist denkbar. In einer nahezu persistenten Welt kann bei einem Statuswechsel ebenfalls ausgegangen werden, dass das verschwundene Objekt in sehr kurzer Zeit ersetzt wird und die Zellen wieder diesen Status erhalten. Daher muss die Frage nach der Umgebung weiter spezifiziert werden, dem Anwendungsszenario. Mögliche Szenarien können je nach Einsatzgebiet des autonomen Systems Baustellen, Wohnkomplexe, Bürogebäude oder auch im Outdoor-Bereich liegen. Viele Szenarien lassen sich nicht in einen klar definierten Dynamikumfang beschreiben. Sie treten nur zu unterschiedlichen Wahrscheinlichkeiten, woraufhin jedes Szenario einen Hybriden zwischen volatilen und persistenten Elementen mit allen Variationen darstellt. Ein weiterer wichtiger Faktor ist das Objekt an sich, woraus viele Informationen in Bezug auf das dynamische Verhalten abgeleitet werden können. Aktuell behandelt der Algorithmus jede Zelle unabhängig, nur mit Informationen über den Ort und der Zeit. Aus diesen Gründen gibt es keine triviale Funktionen. Die im Kapitel IV auf Seite 26 verwendeten Funktionen wurden aus diesen Gründen verhältnismäßig einfach gehalten.

IV. UMSETZUNG & ERGEBNISSE

A. Entwicklungs- und Testumgebung / Komponenten

Das Software-Framework **ROS** (Roboter Operating System [14]) beinhaltet für unterstützte Roboter alle benötigten Softwarepakete, um eine einfache Benutzung von Robotern zu gewährleisten. Hierzu zählt auch eine Simulationsumgebung, **Gazebo** [15], in der virtuelle Areale generiert werden können. Mit dem Softwaremodul **Rviz** [16] lassen sich eine Vielzahl von Komponenten visualisieren, z. B. Sensordaten, Robotermodelle, Transformationen, Planungswege, die Umgebungskarte, uvm, sowie

```

1 Entscheidungsfindung ( $z_{combined}$ ,  $M_{time\ occupied}$ ,  $M_{time\ free}$ ,  $M_{time\ unknown}$ ,  $threshold_{free}$ ,  $threshold_{occupied}$ )
2 for  $i=1$  to number of elements in vec $z_{combined}$  do
3   if currentState==occupied then
4     |  $M_{time\ free}(z_{combined}(x_i, y_i)) = 0$ 
5   end
6   else
7     | timestamp_occupied= $M_{time\ occupied}(z_{combined}(x_i, y_i))$ 
8     | timestamp_free= $M_{time\ free}(z_{combined}(x_i, y_i))$ 
9     | if timestamp_occupied==0 then
10       |   |  $M_{time\ occupied}(z_{combined}(x_i, y_i)) = \text{Timestamp:Now}$ 
11       |   exit
12     | else if timestamp_free==0 then
13       |   |  $M_{time\ free}(z_{combined}(x_i, y_i)) = \text{Timestamp:Now}$ 
14       |   exit
15     | end
16     | dT_occupied = Timestamp:Now - timestamp_occupied
17     | dT_free = Timestamp:Now - timestamp_free
18     | if  $dT_{occupied} > dT_{free}$  then
19       |   | if  $g_{free}(dT_{occupied}, dT_{free}) > threshold_{occupied}$  then
20         |   |   |  $M_{global}(z_{combined}(x_i, y_i)) = \text{Frei}$ 
21         |   |   |  $M_{time\ occupied}(z_{combined}(x_i, y_i)) = 0$ 
22       |   | else
23         |   |   | if  $g_{occupied}(dT_{occupied}, dT_{free}) > threshold_{free}$  then
24           |   |   |   |  $M_{global}(z_{combined}(x_i, y_i)) = \text{Besetzt}$ 
25           |   |   |   |  $M_{time\ free}(z_{combined}(x_i, y_i)) = 0$ 
26         |   |   end
27       |   end
28     end
29 end

```

TABLE XIII: Algorithmus zur Entscheidungsfindung.

Bewegungsbefehle (z. B. Zielpunkte, die angefahren werden sollen) an den Roboter übergeben. Durch die Kombination von Gazebo und Rviz ist es möglich, das Verhalten von Robotern auf Befehle zu analysieren und Softwaremodule direkt ohne Hardware zu testen. Das im Forschungsprojekt verwendete autonome System ist ein Care-O-Bot 3.9, entwickelt vom Fraunhofer-Institut für Produktionstechnik und Automatisierung [17]. Hierfür werden bereits Softwarepakete bereitgestellt, die während des Projekts verwendet werden:

- Das Robotermodell Care-O-Bot 3.9 für Rviz und Gazebo
- Eine virtuelles Areal in Gazebo, „IPA-Appartement“
- Konfiguration für die Visualisierung von Komponenten in Rviz
- SLAM-Algorithmus für die erstmalige Erstellung der Umgebungskarte

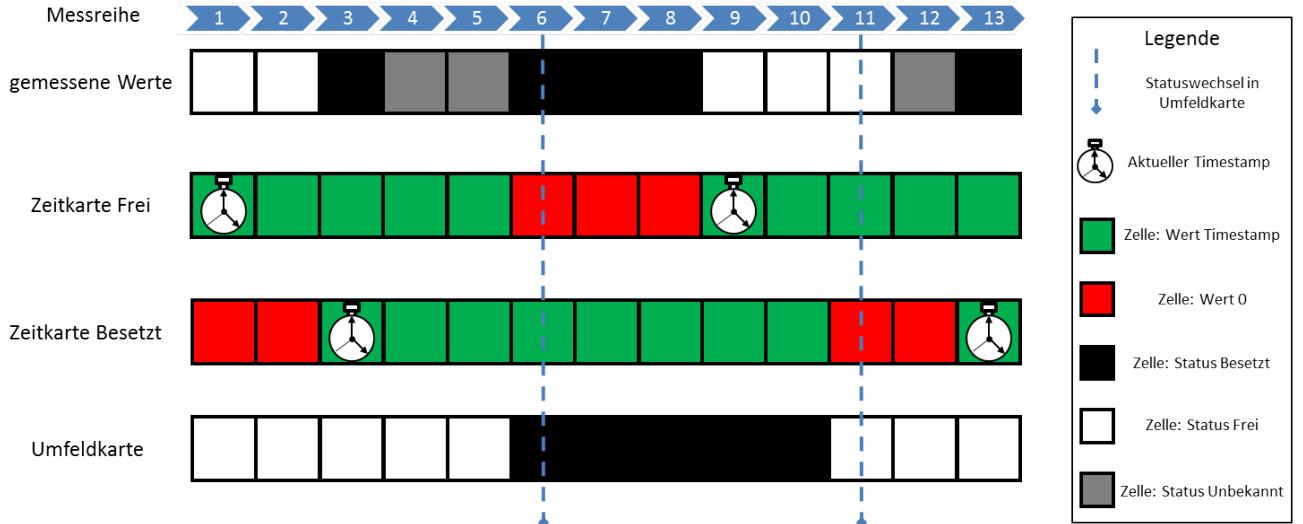


Fig. 16: Verhalten der Informationen in den verschiedenen Karten. Die Zelle wird mit dem Status Frei initialisiert (Messreihe 1). Ist eine Differenz der Status vorhanden, gemessener Wert \neq Umfeldkarte, wird die aktuelle Zeit in der Zelle der entsprechenden Karte gespeichert. In diesem Fall, da der Wechsel von Frei auf Besetzt vollzogen werden soll, liegt der Zeitstempel in der Zeitkarte Besetzt (Messreihe 3). In diesem Beispiel wird der Status in der Umfeldkarte geändert, wenn zwei weitere Messreihen durchlaufen werden. Der Status Unbekannt ändert den Inhalt der Zeitkarten nicht, der Statuswechsel wird aber erst durchgeführt, sobald der neue mögliche Status, hier Besetzt, wieder eintritt. So wechselt der Status in 6 von Frei auf Besetzt, in der Zeitkarte Frei wird der Wert 0 eingetragen. In 9 wird ein neuer möglicher Statuswechsel erkannt, Zeitkarte Frei erhält einen Zeitstempel. Nach zwei Schritten auf 11 wird in Zeitkarte Besetzt der Inhalt auf 0 gesetzt und die Zelle Umfeldkarte wechselt von Besetzt auf Frei. 13 besitzt wieder ein gültiges Messsignal, Zeitkarte Besetzt erhält einen Zeitstempel.

- globaler und lokaler Pfadplaner
- Lokalisierungsalgorithmus für die Bestimmung der Pose des Roboters auf der Umgebungskarte

Die erste Testreihe beschäftigt sich mit der Validierung des Algorithmus in Bezug auf der in Figure 16, S. 28 vorgestellten Funktionsweise. Hierbei wurden die Bedingungen

$$g_{free}(dT_{occupied}, dT_{free}) > threshold_{occupied}$$

sowie

$$g_{occupied}(dT_{occupied}, dT_{free}) > threshold_{free}$$

aus Table XIII auf Seite 27, Zeile 18 und 23 so gewählt, dass die Zellen der Karte M_{global} ab 15 Sekunden nach Markierung der Zellen als potentielle Kandidaten den Status wechseln. Zum Zeitpunkt t_0 ist das Areal komplett frei von blockierenden Objekten, alle Zellen in der Umgebungskarte haben den Zustand Frei. In t_1 wird ein Objekt in das reale Umfeld gesetzt und von den Sensoren erkannt. Nach Ablauf der 15 Sekunden werden die durch das Objekt blockierten Zellen in der Umgebungskarte auf den Status Besetzt geändert (t_2). Das Objekt wird in t_3 entfernt, in der Umfeldkarte bleiben die Zellen noch besetzt (Figure 17, S. 31).

$t_0 = 0\text{s}$. Initialisierung des Szenarios

Reales Umfeld = Freies Gebiet

Umgebungskarte = Frei

$t_1 = 5\text{s}$. Ein Objekt wird ins reale Gebiet eingesetzt

Reales Umfeld = Besetztes Gebiet

Umgebungskarte = Frei, aber Sensor hat Objekt erkannt

$t_2 = 20\text{s}$. Sensoren haben 15 Sekunden ein besetztes Gebiet erkannt

Reales Umfeld = Besetztes Gebiet

Umgebungskarte = Besetztes Areal wurde in Umgebungskarte integriert

$t_3 = 25\text{s}$. Das Objekt wird aus dem Szenario entfernt

Reales Umfeld = Freies Gebiet

Umgebungskarte = Besetztes Gebiet, Sensoren registrieren freie Fläche

Dieses Szenario zeigt somit, dass die Einbindung der Zeit in die Umgebungskarte funktioniert.

Für die zweite Testreihe wurden die Bedingungen zum Umschalten der Zellstatus auf 5 Sekunden geändert. Zu Beginn des Szenarios wurde eine Umgebungskarte mittels SLAM (GMapping) erstellt. Die physische Umgebung und die daraus erstellte Karte sind in Figure 18, S. 32 sichtbar. Der Pfad ist in diesem Fall durch einen braunen Kasten blockiert (rote Markierung), der Pfad ist demnach nicht nutzbar.

Zu einem späteren Zeitpunkt findet eine Wegplanung mit in Figure 19, S. 33 abgebildeter Umwelt statt. Der Roboter hat hier die Aufgabe Punkt B zu erreichen. Es stehen zwei mögliche Trajektorien zur Verfügung:

- Gelbe Linie: Weg direkt zum Zielpunkt. Nur möglich, wenn dieser Weg nicht blockiert ist.
- Weiße Linie: Umweg, wenn direkter Pfad nicht befahrbar ist.

In diesem Fall ist der gelbe Pfad befahrbar, das blockierende Objekt ist nicht mehr vorhanden. Obwohl die Möglichkeit einer wesentlich kürzeren Strecke besteht, entschied sich der globale Planer für den längeren Weg (Figure 20, S. 34). Durch die erhaltenen Sensorinformationen wurde diese Änderung der physischen Umgebung registriert und die blockierenden Zellen als mögliche Kandidaten für einen Statuswechsel markiert (Figure 21, S. 35). Nach Ablauf der 5 Sekunden wechselten die markierten Zellen in den Zustand Frei, die Umgebungskarte wurde angepasst (Figure 22, S. 36), der Korridor ist auf der Umgebungskarte als befahrbar markiert. Der globale Planer passt sich an die neue Situation an (Figure 23, S. 36), plant die Trajektorie um und nutzt nun den wesentlich effizienteren Weg zum Ziel. Durch die Integration der veränderten Umgebung wird die Effizienz der Pfadplanung in Bezug auf die genutzten Trajektorien verbessert.

V. ZUSAMMENFASSUNG & AUSBLICK

Im Rahmen des Forschungsprojekts wurde die Frage nach einer sinnvollen Adaption der langfristigen dynamischen Prozesse in der Umwelt in ein autonomes System bearbeitet. Die Nutzung der verstrichenen Zeit zwischen verschiedenen erkannten Status wurde hierbei als entscheidender Faktor

für die Lösung der Frage identifiziert. Eine naive Testreihe hat die grundlegende Funktionalität des Ansatzes bestätigen können und zeigt das Potential auf und kann als Grundlage weiterer Forschungen verwendet werden. Die nächsten Schritte könnten sein, dem System das optimale Auswahlverfahren durch maschinelles Lernen selbst beizubringen. Da verschiedene Objekte unterschiedliche Zeitverhalten aufweisen (z. B. Möbel zu Wänden) ist auch eine Gruppierung und Klassifizierung der Objekte als Erweiterung denkbar.

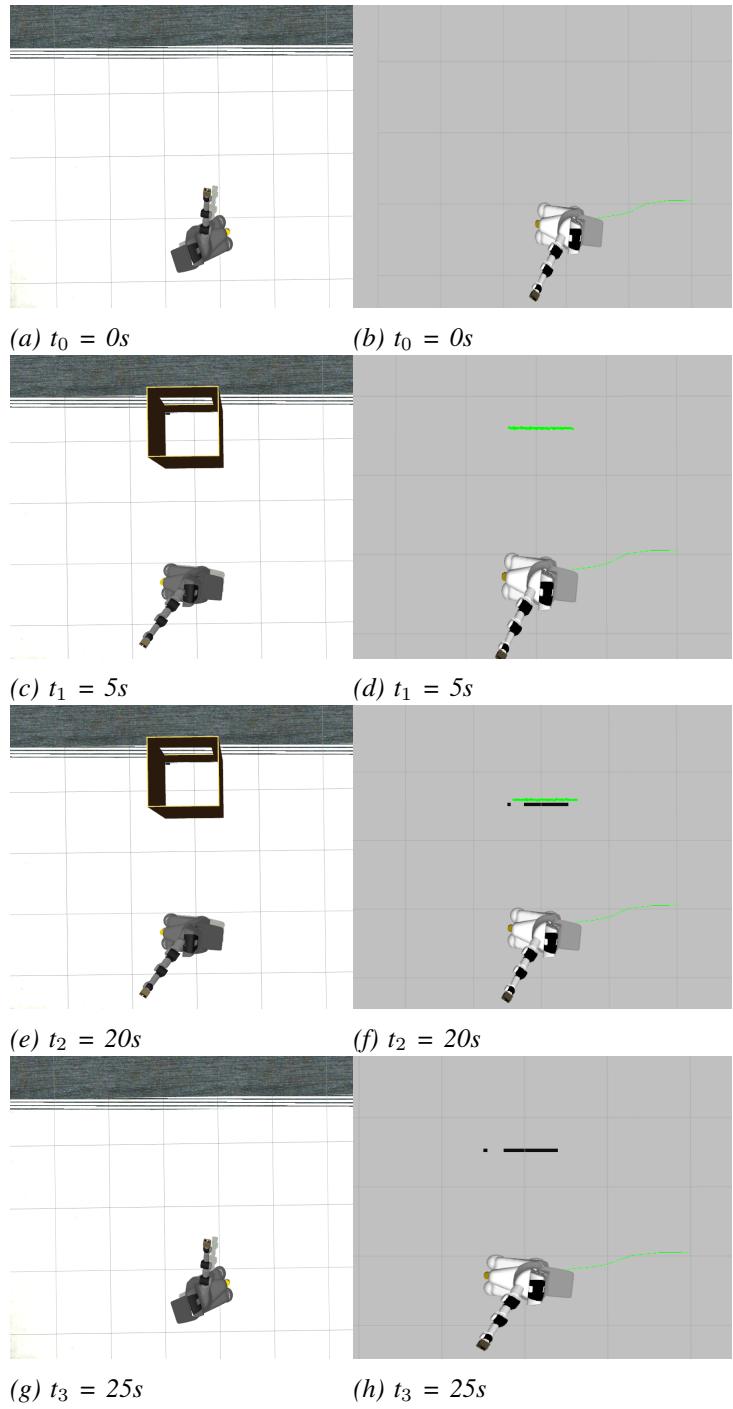
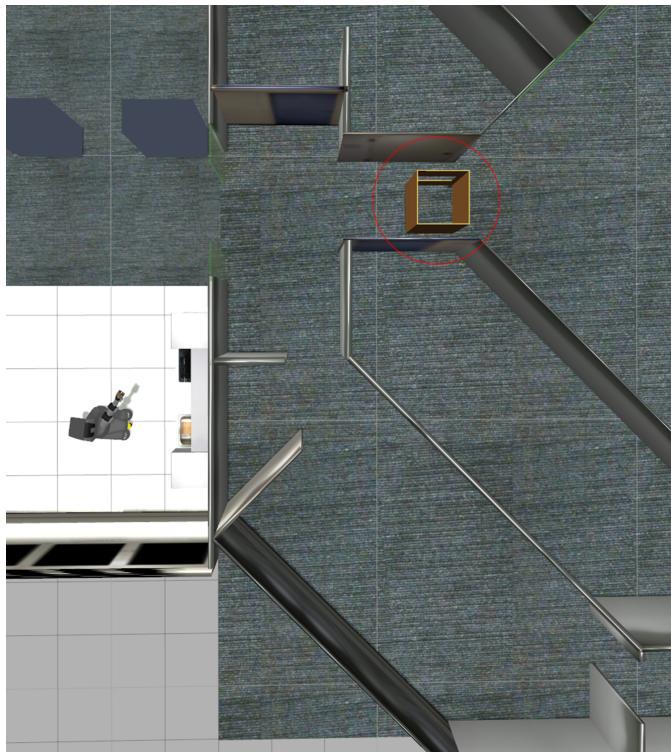
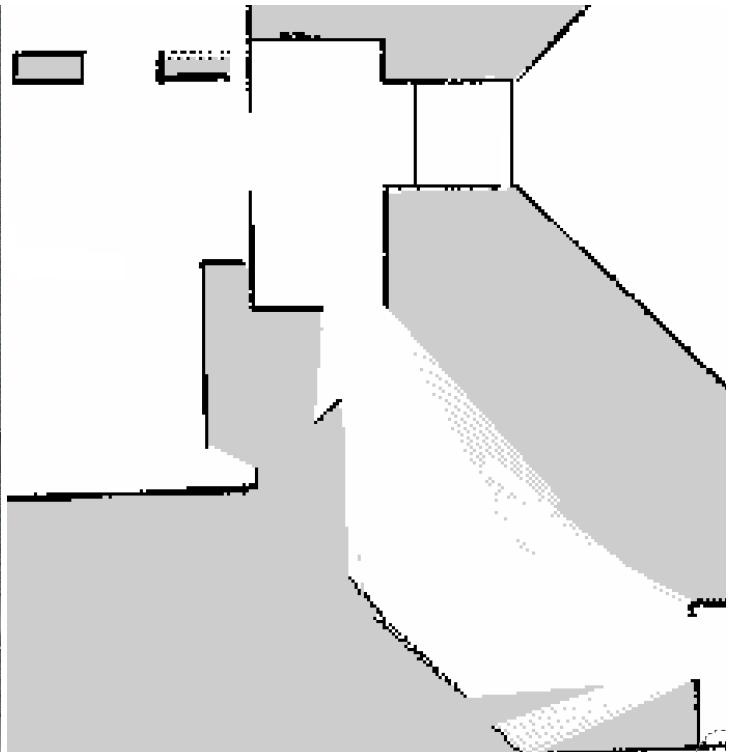


Fig. 17: Adaption der Umfeldkarte M_{global} . Das reale Umfeld ist in (a), (c), (e), (g) zu unterschiedlichen Zeiten, abgebildet. Die daraus resultierende Umfeldkarte M_{global} ist in (b), (d), (f), (h) sichtbar.



(a) Physische Umgebung



(b) Aus der physischen Umgebung resultierende Karte

Fig. 18: Die physische Umgebung des Szenarios (a) sowie die daraus resultierende Umgebungskarte (b). Ein brauner Kasten (Rot umkreist) blockierte zum Zeitpunkt der Kartenerstellung einen Korridor.

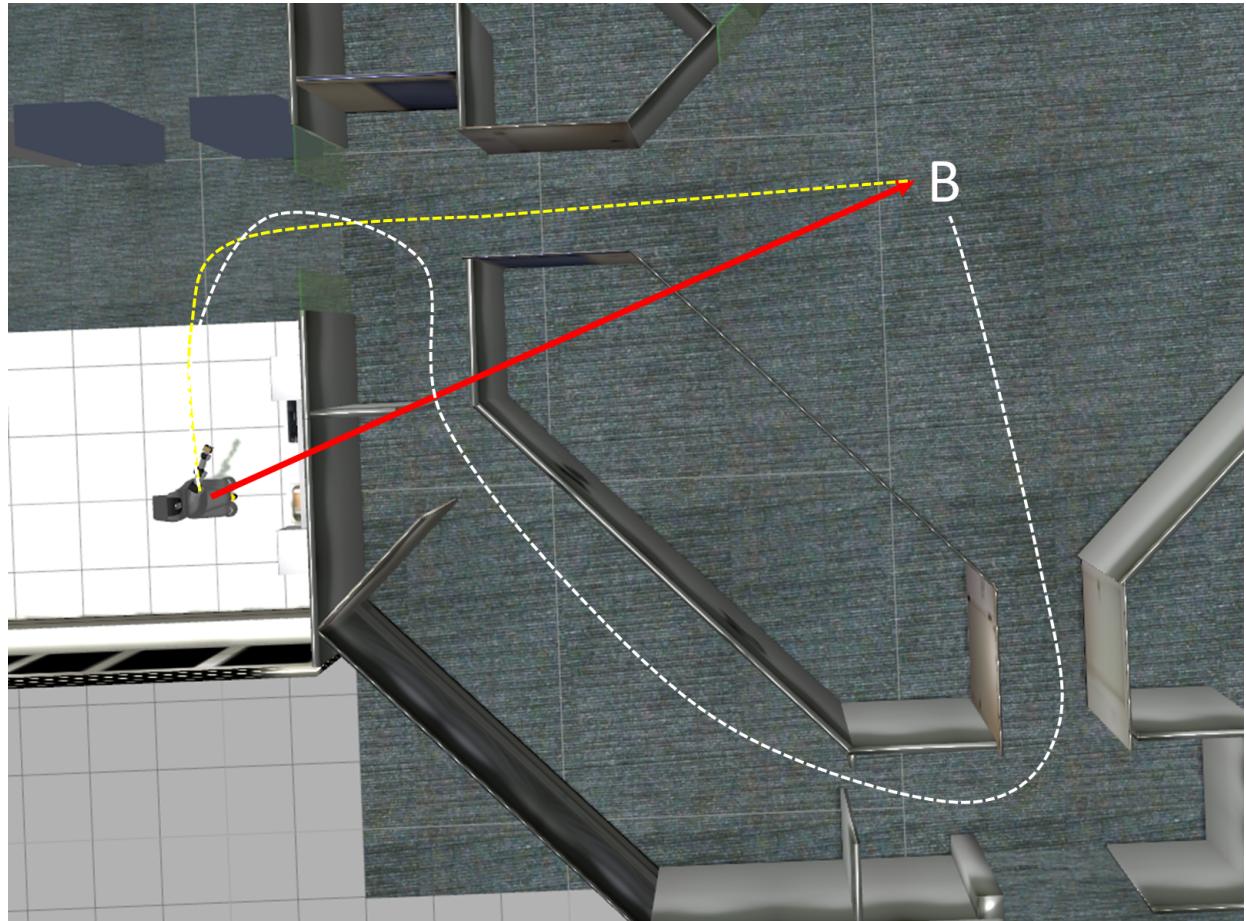
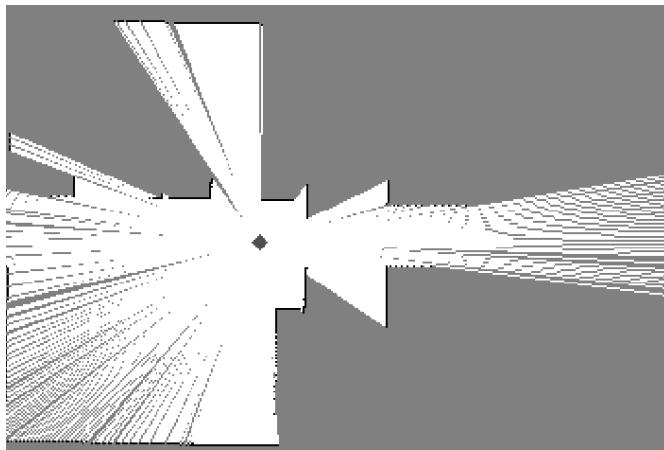


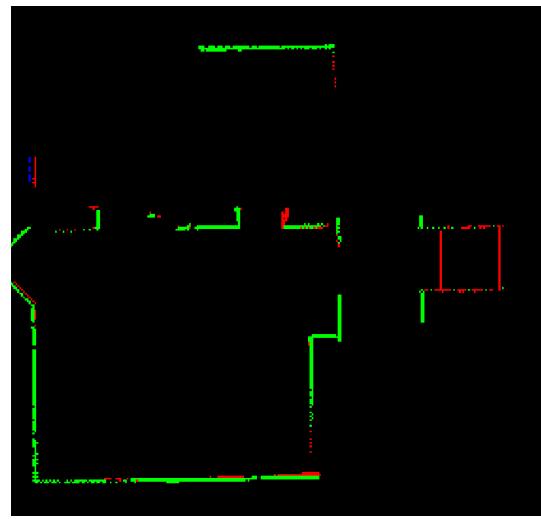
Fig. 19: Zielareal des Szenarios. Aufgabe ist das Erreichen von Punkt B, ausgehend vom Startpunkt. Die beiden Linien (Weiß, Gelb) markieren die möglichen Trajektorien, wobei die gelbe Linie den optimalen Weg markiert.



Fig. 20: Ergebnis der globalen Pfadplanung (grüne Linie) bei Verwendung der Umgebungskarte mit blockiertem Korridor. Es wird ein Alternativpfad berechnet, obwohl die physische Umwelt keine Blockade des Korridors aufweist (Figure 19, S. 33).



(a) Lokale Umgebungskarte M_{lokal} . In diesem Abbild ist kein blockierendes Objekt im Korridor auffindbar.



(b) Grafische Darstellung der Zellenmarkierung für einen Statuswechsel. Grüne Zellen beschreiben keinen nötigen Statuswechsel, $M_{\text{lokal}}(x, y) = M_{\text{global}}(x, y)$. Rote Zellen sind Wechselkandidaten.

Fig. 21: Diese Abbildung zeigt die Sicht der Sensordaten in der aktuellen Umgebung (a). Der offene Korridor wurde erkannt und die in diesem Fall die zu ändernden Zellen in (b) rot markiert.



Fig. 22: Anpassung der Umgebungskarte. Nach Registrierung eines Statuswechsel und Ablauf der eingestellten Zeitgrenze wechselt der blockierte Bereich in den Status Frei. Der Korridor ist somit befahrbar.

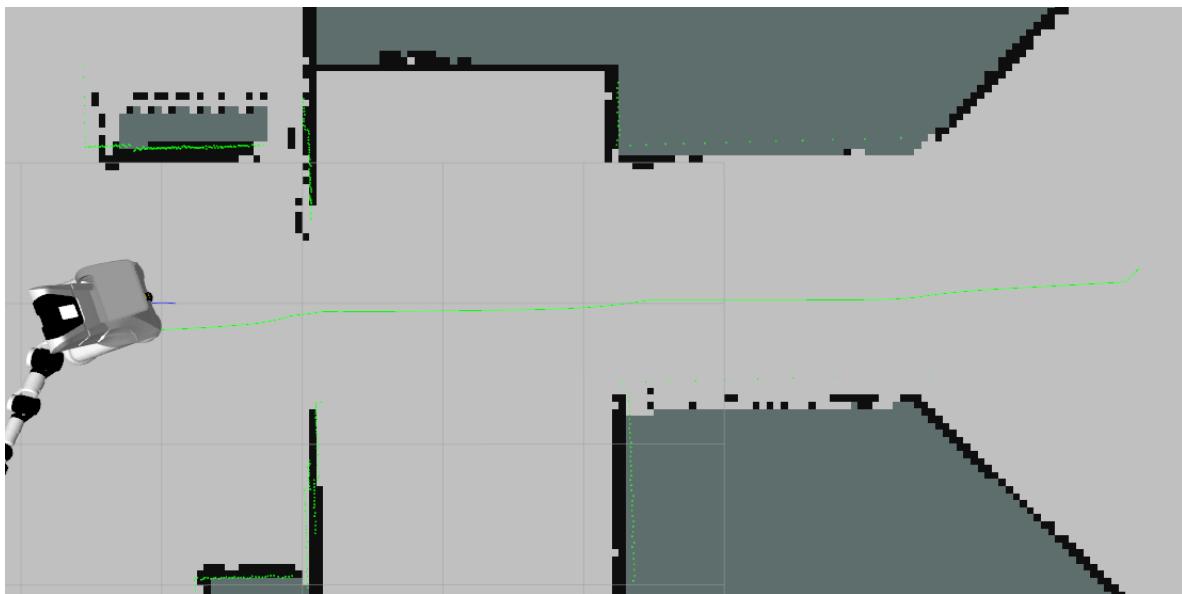


Fig. 23: Der globale Pfadplaner passt sich an die neuen Gegebenheiten der Umgebungskarte an und plant den Pfad durch den neuen Korridor.

REFERENZEN

- [1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard, “Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age,” *IEEE Transactions on Robotics*, vol. 32, no. 6, p. 13091332, 2016.
- [2] J. M. Santos, D. Portugal, and R. P. Rocha, “An evaluation of 2D SLAM techniques available in robot operating system,” in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2013, pp. 1–6.
- [3] P. Biber and T. Duckett, “Dynamic Maps for Long-Term Operation of Mobile Service Robots.” in *Robotics: Science and Systems 2005*, Jan. 2005, pp. 17–24.
- [4] A. Walcott-Bryant, M. Kaess, H. Johannsson, and J. J. Leonard, “Dynamic pose graph SLAM: Long-term mapping in low dynamic environments,” in *Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2012, pp. 1871–1878.
- [5] D. F. Wolf and G. S. Sukhatme, “Towards mapping dynamic in environments,” in *Proceedings of the International Conference on Advanced Robotics (ICAR)*, 2003, pp. 594–600.
- [6] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun, “Towards object mapping in non-stationary environments with mobile robots,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 2002, pp. 1014–1019 vol.1.
- [7] N. C. Mitsou and C. S. Tzafestas, “Temporal occupancy grid for mobile robot dynamic environment mapping,” in *Mediterranean Conference on Control & Automation, 2007*. IEEE, 2007, pp. 1–8.
- [8] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, Aug. 2005.
- [9] “Amcl - ROS Wiki,” <http://wiki.ros.org/amcl>, online; accessed 05-01-2016.
- [10] “Nav_core - ROS Wiki,” http://wiki.ros.org/nav_core, online; accessed 23-01-2016.
- [11] “Move_base - ROS Wiki,” http://wiki.ros.org/move_base, online; accessed 1-02-2016.
- [12] G. Grisetti, C. Stachniss, and W. Burgard, “Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005, pp. 2432–2437.
- [13] “Sensor_msgs/LaserScan Documentation,” http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html, online; accessed 30-01-2016.
- [14] “Ros - Robot Operating System,” <http://www.ros.org/>, online; accessed 29-12-2016.
- [15] “Gazebo - Robot simulation made easy,” <http://gazebosim.org/>, online; accessed 29-12-2016.
- [16] “Rviz - 3D visualization tool for ROS,” <http://wiki.ros.org/rviz>, online; accessed 29-12-2016.
- [17] “Care-O-bot 3,” <http://www.care-o-bot.de/de/care-o-bot-3.html>, online; accessed 31-12-2016.