Autonome Klassifizierung, Planung und Navigation auf schwer zugänglichem Terrain

Diplomarbeit an der Albert-Ludwigs-Universität Freiburg

Fakultät für Angewandte Wissenschaften

vorgelegt von:

Christian Dornhege

Erstgutachter: Prof. Dr. Bernhard Nebel
Zweitgutachter: Prof. Dr. Wolfram Burgard
Betreuer: Alexander Kleiner

2007

Zusammenfassung

Ziel dieser Diplomarbeit ist es ein Verfahren zur Exploration auf schwer zugänglichem Terrain, das Hindernisse wie Paletten und Rampen beinhaltet, zu entwickeln. Im Vergleich zu Umgebungen wie zum Beispiel Schotter oder Gras, die auch erhöhte Anforderungen an die Mobilität des Roboters stellen, ist es bei sehr steilen Strukturen nicht mehr möglich nur zwischen befahrbar und Hindernis zu unterscheiden und das Vorankommen des Roboters durch ein einziges Verhalten zu gestalten. Es ist vielmehr notwendig, dass der Roboter sich der spezifischen Hindernisse bewusst ist und dafür spezielle Fähigkeiten besitzt und ausführen kann. Um Hindernisse überhaupt zu erkennen, werden deshalb während der Fahrt Höhenkarten mit einem geneigten 2D-Laserscanner erstellt und diese dann klassifiziert. Aus Höhenkarten und einer Menge von Fähigkeitsbeschreibungen für verschiedene Hindernisse werden dann automatisch *Verhaltenskarten* erstellt. Diese kodieren mittels der Fähigkeitsbeschreibungen direkt Informationen für die auszuführende Fähigkeitsroutine in die Karte. Dadurch wird die Planung mittels normaler A*-Suche unter Berücksichtigung der spezifischen Voraussetzungen zur Überquerung von Hindernissen ermöglicht. Experimente, die unter anderem einen vollständig autonomen Lauf in einem Testparcours, bei dem eine Palette und eine Rampe zu überqueren war und eine Höhenkarte erstellt wurde, beinhalten, demonstrieren die Möglichkeiten des entwickelten Systems.

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Christian Dornhege
Freiburg, den 10. April 2007

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Zielsetzung

Das Überwinden von schwierigem Terrain ist eine große Herausforderung für autonome Roboter, die sowohl geeignete Algorithmen, als auch eine Robotikbasis mit hoher Mobilität erfordert. Viele Ansätze beschäftigen sich daher mit der Klassifikation in befahrbare Strukturen und nicht befahrbare Strukturen oder mit der Erkennung von mäßig schwerem Terrain, wie Schotter, Sand oder Gras [18, 20]. Selbst gewohnte menschliche Umgebungen enthalten allerdings für Roboter schwierig zu überwindende Hindernisse wie Treppen und Bordsteinkanten, die für solche Vorgehensweisen ein Problem darstellen. Gerade im Bereich Urban Search and Rescue (USAR) ist das Vermeiden von Hindernissen allein nicht sehr erfolgsversprechend, da oftmals der einzig mögliche Weg über ein Hindernis führt. Teleoperierte Rettungsroboter, wie unter anderem auch die Lurker-Roboter des *RescueRobots Freiburg*-Teams [14], die bei den Robocup-Wettbewerben eingesetzt werden, besitzen die Fähigkeit verschiedene Hindernisse zu überwinden. Da solche Roboter zunehmend auch autonom eingesetzt werden [13], ist es naheliegend die ferngesteuerten Fähigkeiten im Bereich der Mobilität auch autonom umzusetzen.

Schwer zugängliches Terrain, das steile Strukturen erhält, macht es allerdings notwendig, dass der Roboter sich seines Kontexts explizit bewusst ist, um so situationsspezifische Verhalten ausführen zu können. Grundsätzlich erfordert dies erst einmal, dass der Roboter ein geeignetes Weltmodell besitzt, das die Strukturen mit hinreichender Genauigkeit abbilden kann und in diesem Kontext auch das gleichzeitige Erstellen von Karten und Lokalisieren des Roboters, dass als *Simultaneous Localization And Mapping* (SLAM) bekannt ist. Des weiteren müssen diese Daten interpretiert werden, um so ein Weltverständnis zu erlangen, dass es erlaubt geeignete Pläne zu erzeugen und auszuführen. Außerdem ist es natürlich notwendig autonome Verhalten zu besitzen, die den Roboter überhaupt über verschiedenartige Hindernisse steuern können. In einer früheren Arbeit [25] wurden die Roboter schon mit entsprechenden Sensoren ausgestattet und eine Reihe von Spezialfähigkeiten im-

plementiert, die den Roboter Paletten, Rampen und Treppen überwinden lassen. Es stellt sich nun also die Aufgabe ein Gesamtsystem zu entwickeln, das den Roboter befähigt eine dreidimensionalen Umgebung wahrzunehmen, eventuell vorhandene Hindernisse zu erkennen, Pfade über diese Hindernisse zu planen und die Welt letzten Endes autonom zu explorieren.

Diese Arbeit beschäftigt sich deshalb im ersten Teil mit einer geeigneten Weltmodellierung. Dazu sollen Höhenkarten [3, 21] verwendet werden, die während der Fahrt mit einem geneigten Laserscanner in Echtzeit erzeugt werden. Die gleichzeitige Lokalisierung in drei Dimensionen wird mit einem erweiterten Kalman-Filter vorgenommen, dessen Eingaben hauptsächlich von einem in zwei Dimensionen arbeitenden SLAM-Algorithmus stammen, der einen zweiten horizontal befestigten Laserscanner benutzt. Zur Abschätzung der Roboterhöhe werden des weiteren der Nickwinkel eines Lagesensors und Messungen der aktuellen Roboterhöhe aufgrund der Karte mit den Posen des 2D-SLAM-Algorithmus zu einer dreidimensionalen Pose verschmolzen. Aus den so erzeugten Höhenkarten werden mittels einer allgemeinen Formulierung zur Integration von Spezialfähigkeiten durch *Fähigkeitsbeschreibungen* dann automatisch *Verhaltenskarten* erstellt. Verhaltenskarten sind zellenbasierte Karten, die die Planung und Planausführung unter Berücksichtigung der Einschränkungen durch die Umgebung erlauben. Dabei bilden klassifizierte Zellen direkt auf *Verhaltensroutinen* ab und erlauben so die kontextspezifische Ausführung von verschiedenen Roboterfähigkeiten.

Die verschiedenen *Fähigkeitsbeschreibungen* enthalten dabei Fuzzy-Regeln, die für die Klassifikation des jeweiligen Hindernisses benötigt werden, eine Menge von räumlichen Beziehungen, eine Kostenfunktion für die Planung und eine Verhaltensroutine, die das entsprechende Verhalten ausführt. Die Fuzzy-Regeln werden dabei zur Erzeugung von Merkmalen benutzt, die der Klassifikation von Höhenkarten mit Hilfe von *Markov-Random-Fields* dienen. Die räumlichen Beziehungen kodieren Übergänge zwischen verschiedenen Regionen, die das Verhalten überwinden kann, und werden dazu benutzt aus den Übergangskanten Vorabbedingungen, wie Startposition und Startrichtung, die zur Ausführung des Verhaltens notwendig sind, zu bestimmen. Die Planung auf Verhaltenskarten ist dann durch einem normalen A*-Suchalgorithmus möglich, der die spezifischen Kostenfunktionen berücksichtigt.

Der Rest dieser Arbeit ist wie folgt strukturiert: Zunächst wird auf verwandte Arbeiten eingegangen und in Abschnitt 1.3 die verwendete Hardware vorgestellt. In Kapitel 2 werden einige Definitionen und bekannte Algorithmen eingeführt, die in dieser Arbeit benutzt werden. Kapitel 3 beschäftigt sich mit der Weltmodellierung mittels Höhenkarten und in Kapitel 4 wird erläutert wie *Verhaltenskarten* erzeugt werden und die Planung auf diesen erfolgt. Kapitel 5 stellt die Ergebnisse aus Experimenten mit den *Lurker*-Robotern dar und Kapitel 6 fasst die Arbeit noch einmal zusammen und gibt einen Überblick über mögliche Erweiterungen des Systems.

1.2 Verwandte Arbeiten

Um dreidimensionale Karten zu bauen, benutzen *Thrun et al.* [28] einen zweidimensionalen Lokalisierungsalgorithmus, um mit einem zweiten um 90° geneigten Laserscanner Höhenstrukturen wahrzunehmen. Die Verwendung von Höhenkarten als kompakte Repräsentation statt Punktwolken oder Polygonen kommt in vielen Anwendungen zum Einsatz. So bauen *Pfaff et al.* [21] aus 3D-Scans Höhenkarten, die mittels ICP zu einer Karte verschmolzen werden, und erweitern das Verfahren, so dass vertikale Strukturen und Überhänge wie Brücken erkannt und in Hinblick auf die Roboternavigation besser abgebildet werden. Dieses Prinzip wird von *Triebel et al.* [30] verallgemeinert, die die Definition von Höhenkarten zu Multi-Level-Surface-Maps erweitern, welche verschiedene Ebenen und vertikale Strukturen abbilden können. Um während der Roboterfahrt Höhenkarten erstellen zu können verwenden *Ye* und *Borenstein* [36] einen zum Boden geneigten Laserscanner aus dessen Entfernungsmessungen mit der aktuellen Pose Höhenkarten erstellt werden. Um durch den Sensor verursachte Fehler zu unterdrücken, wird dabei ein adaptiver Filteralgorithmus benutzt, der die Karte gezielt an unsicheren Stellen durch einen Medianfilter verbessern kann.

Algorithmen zur autonomen Fahrt auf schwierigem Terrain sind häufig verhaltensbasierte Ansätze, die sich deshalb auch mit der Erkennung von Terrain wie Asphalt, Sand, Gras und Schotter befassen. *Ye* und *Borenstein* [38] benutzen dazu die T-Transformation, bei der für die Zellen einer Höhenkarte ein “Trafficability Index” bestimmt wird, der auf der Neigung einer lokalen gefitteten Ebene und der dabei ermittelten Varianz beruht. Dieser Ansatz kommt dann auf einem Roboter auf Segway-Basis [37], der mit einem geneigten Laserscanner Höhenkarten erzeugt, zum Einsatz. Einen vergleichbaren Roboter verwenden *Wolf et al.* [34], die dann direkt aus den Laserscans mit einem Hidden-Markov-Model befahrbare Flächen erkennen, und so zum Beispiel Gras umfahren können. Basierend auf Parametern, die während der Roboterfahrt mit einem Gyroskop, sowie Sensoren, die den Motorstrom messen, ermittelt werden, unterscheiden *Ojeda et al.* [20] Terrain wie Sand, Gras und Asphalt, um so das Fahrverhalten des Roboters anzupassen. Der Roboter Stanley, der von *Thrun et al.* [29] entwickelt wurde, benutzt sowohl Laserscanner als auch Kameradaten, um befahrbares Terrain zu ermitteln. Außerdem werden roboterinterne Sensoren wie der Nickwinkel und die z-Beschleunigung verwendet, um das Fahrverhalten adaptiv der Umgebung anzupassen. *Waldron et al.* [33] benutzen ein Robotermodell, das eine aktive Aufhängung besitzt, womit sich Räder heben und senken lassen. Darauf aufbauend wird ein Verfahren beschrieben, mit dem es möglich ist, durch gezielte Veränderung der aktuellen Roboterkonfiguration verschiedene geometrische Hindernisse zu überwinden. *Hebert et al.* [9] stellen verschiedene Fahrmodi für ihren Roboter vor, die sich in städtischen Umgebungen semiautonom

fortbewegen und dabei auch Hindernisse überwinden können. Zur Navigation wird dazu Visual Servoing eingesetzt, das ein vom Operator ausgewähltes Ziel oder eine Reihe von Wegpunkten anfahren kann.

Mit der Klassifikation von Terrain befassen sich auch *Vandapel et al.* [31]. Mittels einer 3D-Ladar Sensors werden dabei Punktwolken auf der Basis von lokalen Statistiken untersucht. Dabei wird das Verhältnis der Eigenwerte der Kovarianzmatrix, ähnlich dem Verfahren zur Merkmalsauswahl beim KLT-Tracking [17, 23], untersucht, um so Ebenen (ein dominanter Eigenwert), lineare Strukturen wie Drähte (zwei dominante Eigenwerte) und strukturlose Daten, wie Gebüsch (kein dominanter Eigenwert) unterscheiden zu können. Der Ansatz von *Talukder et al.* [26] versucht nicht nur Objekte an sich zu erkennen, sondern auch deren Konsistenz. Dazu werden sowohl Farb- und Texturmerkmale, sowie auch geometrische Parameter beachtet. Damit soll zum Beispiel erkannt werden, dass sich Gras und Gebüsch beim Darüberfahren herunterdrückt und somit kein wirkliches Hindernis darstellt. Diese Informationen werden unter Beachtung des Federungsmodells des Roboters dazu benutzt Fahrparameter terrainspezifisch zu wählen. Das gleiche Problem versuchen *Manduchi et al.* [18] mit einer Farbstereokamera zu lösen. Um in Gras verborgene Steine oder Baumstümpfe zu detektieren, wird zusätzlich ein 2D-Ladar verwendet. Diese Hindernisse werden durch Unregelmäßigkeiten im Histogramm der gemessenen Entfernungen festgestellt.

1.3 Verwendete Hardware

1.3.1 Lurker-Roboter

Als Robotik-Platform werden die auf dem *Tarantula*-Spielzeug basierenden Roboter des *RescueRobots Freiburg*-Teams benutzt, die auch bei den Robocup Weltmeisterschaften in Osaka und Bremen zum Einsatz kamen. Die Roboter verfügen über einen Kettenantrieb und vorne und hinten über frei drehbare Arme, mit denen sie sich auf und über Hindernisse heben können und so die Fähigkeit besitzen schwieriges Terrain, wie Paletten, Rampen und Treppen zu befahren. Die Roboter wurden im Laufe der Entwicklung des *RescueRobots Freiburg*-Teams stark modifiziert. In dem Aufbau des Roboters befindet sich ein Sony Vaio PCG-C1VE Subnotebook, dass die Sensordatenaufnahme über den Mikrocontroller, sowie andere Sensoren bewerkstelligt und für die Ausführung der Verhalten benutzt wird. An den Armachsen befinden sich Potentiometer (Abbildung 1.1(c)), die die Winkelposition der Arme wiedergeben können und in den Armen sind Tastsensoren angebracht, so dass sich der Kontakt eines Armes an einer bestimmten Stelle feststellen lässt. Diese Sensoren ermöglichen das Erklimmen von Hindernissen wie Paletten und Treppen.

1.3.2 XSens Lagesensor (Inertial-Measurement-Unit)

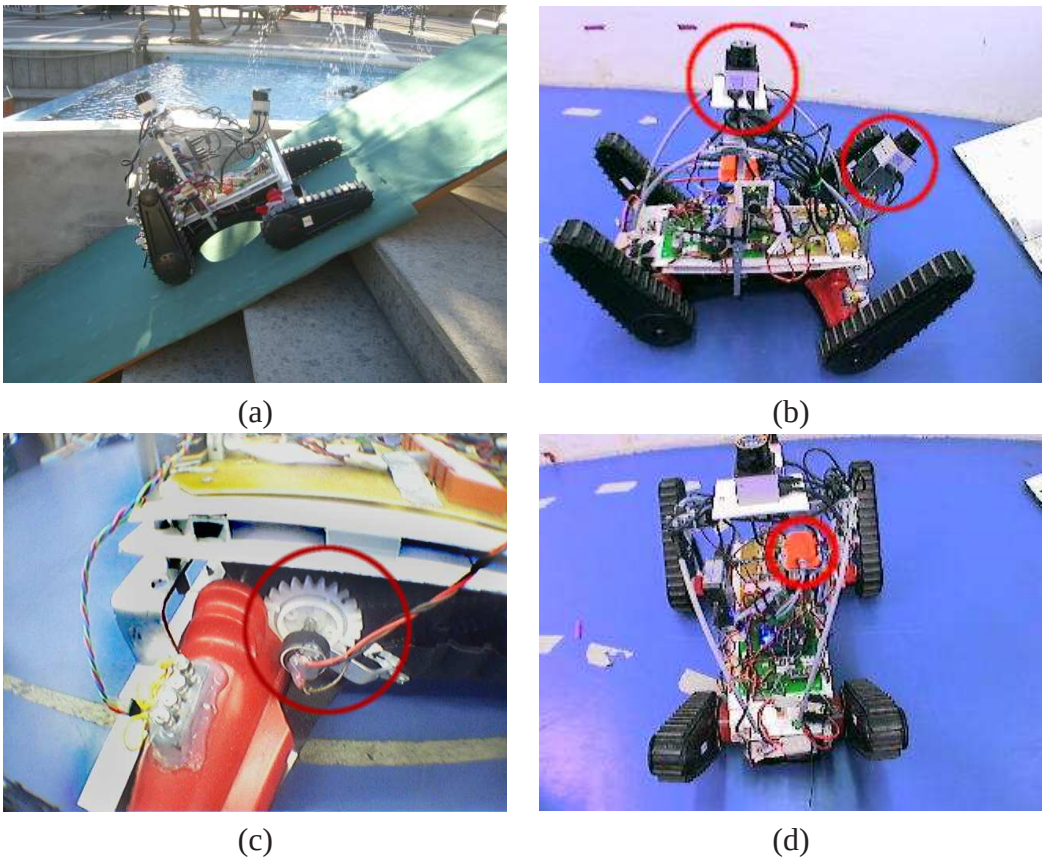
Des weiteren wird ein Lagesensor der Firma XSens eingesetzt, der driftfreie Winkel in allen drei Raumrichtungen mit sehr guter Präzision bereitstellt. Der Sensor ist in Abbildung 1.1(d) zu sehen. Dieser Sensor ist insbesondere für die dreidimensionale Weltmodellierung, sowie für die autonomen Spezialverhalten sehr wichtig.

1.3.3 Hokuyo URG-X004 Laserscanner

Zur Weltwahrnehmung kommen 2D-Laserscanner der Firma Hokuyo zum Einsatz. Diese sind in Abbildung 1.1(b) markiert. Die Sensoren liefern ein zweidimensionales Entfernungsbild in der Sensorebene mit 683 Strahlen über ein Sichtfeld von $240°$. Sie zeichnen sich durch kleine Baugröße, ein geringes Gewicht und niedrigen Stromverbrauch aus, so dass auf dem Roboter zwei dieser Sensoren verwendet werden können. Einer der Sensoren ist dazu oben mittig auf dem Roboter montiert und dient zum Sammeln von 2D-Laserscans für die Lokalisierung. Der zweite Scanner ist auf einer Schwenkvorrichtung montiert und wird in der Regel $-35°$ nach unten geschwenkt betrieben, so dass ein Abscannen der Oberfläche ermöglicht wird und somit ein dreidimensionales Weltbild zustande kommt. Außerdem ist das Erstellen von 3D-Scans durch das Bewegen der Schwenkvorrichtung über den gesamten Winkelbereich von $130°$ möglich.

1.3.4 Logitech QuickCam 4000 Pro

Auf dem Roboter sind außerdem Webcams der Marke Logitech QuickCam 4000 Pro angebracht, die zur Fernsteuerung eingesetzt werden können. Eine seitlich angebrachte Kamera wird für visuelle Odometrie verwandt [6].



(a) (b)

(c) (d)

Abbildung 1.1. Verwendete Hardware: (a) Der Lurker-Roboter. (b) Die zwei Laserscanner für die Lokalisierung (oben) und zum Bauen von Höhenkarten (vorne, geneigt). (c) Potentiometer, das als Armpositionssensor verwendet wird. (d) Der Xsens Lagesensor.

Kapitel 2

Grundlagen

Da diese Diplomarbeit das Ziel hat ein autonomes Robotiksystem mit allen dazu notwendigen Komponenten zu entwickeln, ist es nötig verschiedene Verfahren aus der Robotik und anderen Bereichen zu benutzen, um die vielfältigen Aufgaben zu lösen. In diesem Kapitel sollen daher einige grundlegende Definitionen und Algorithmen vorgestellt werden, die im Laufe der Diplomarbeit zur Anwendung kommen. Auf deren Verwendung wird dann in den Kapiteln 3 und 4 eingegangen.

2.1 Koordinatensystem

Das Roboterkoordinatensystem, mit dem gearbeitet wird, ist in Abbildung 2.1 dargestellt und legt die (xz)-Ebene parallel zum Boden, so dass die x-Achse nach rechts, die y-Achse nach oben und die z-Achse nach hinten zeigt. Es soll deshalb in dieser Diplomarbeit als (xz)-Koordinatensystem bezeichnet werden. Rotationen werden mathematisch positiv entgegen dem Uhrzeigersinn vorgenommen und $R_x(\alpha)$ bezeichnet die Rotationsmatrix mit Winkel α um die x-Achse, sowie $R_y(\alpha)$ und $R_z(\alpha)$ Rotationen um die y-, beziehungsweise z-Achse. Der Ursprung befindet sich dabei in der Mitte des Roboters auf Bodenhöhe.

2.2 Kalman-Filter

Die Aufgabe des Kalman-Filters ist es eine nicht direkt beobachtbare Variable x aufgrund von Messungen z zu schätzen. Dabei wird zwischen Vorhersage und Messung unterschieden. Im Vorhersage-Schritt wird die Variable propagiert, zum Beispiel durch Odometriemessungen eines Roboters. Im Mess-Schritt wird eine Messung, die ein Indiz für die Variable ist, mit der Variable verschmolzen, um so zu einer besseren Einschätzung zu kommen. Es werden dabei zwei grundsätzliche Annahmen getroffen.

- Die Variablenpropagationen und Messungen sind linear.

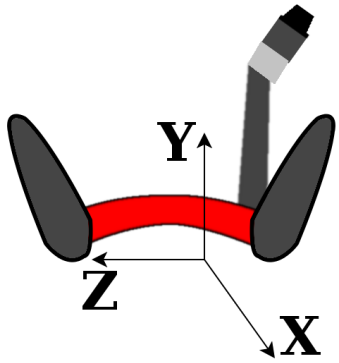


Abbildung 2.1. Das verwendete Koordinatensystem.

- Die Fehler, die mit Propagationen und Messungen verbunden sind, sind normalverteilt.

Der Kalman-Filter nimmt dabei an, dass die geschätzte Variable $\hat{x}$ ebenso normalverteilt mit Kovarianz Σ ist und die Anwendung von Propagation und Messung erhält diese Eigenschaft. Für Systeme, die diese Annahmen erfüllen, liefert der Kalman-Filter die optimale Schätzung für die Variable x.

2.2.1 Der Lineare Kalman-Filter

Der lineare Kalman-Filter nimmt folgendes Modell für die Propagation von x (Gleichung 2.1) und die Messung z (Gleichung 2.2) an, wobei ε und ω Gauss-verteilte von x unabhängige Zufallsvariablen mit Mittelwert null und Kovarianzmatrix Q, beziehungsweise R sind, sowie A, B, H lineare Matrizen und u ein Eingabevektor für eventuelle Steuerwerte ist.

$$x = Ax + Bu + \varepsilon \tag{2.1}$$

$$z = Hx + \omega \tag{2.2}$$

Sind diese beiden Gleichungen bekannt, weiß man also, wie sich das System verhält. Außerdem bekommt man Messungen z aus dem System, so dass es nun möglich ist eine Schätzung für x mittels den Gleichungen 2.3 bis 2.6 zu erhalten [12, 19, 22].

Vorhersage-Schritt

Da ε einen Mittelwert von null hat, folgt direkt die Gleichung für die neue Schätzung von x.

$$\hat{x}_{neu} = A\hat{x} + Bu \tag{2.3}$$

Die Kovarianz wird im Vorhersage-Schritt mit der Propagationsmatrix transformiert und es wird der entsprechende Fehler Q dazugezogen.

$$\Sigma_{neu} = A\Sigma A^T + Q \tag{2.4}$$

Mess-Schritt

Die gemäß dem Kalman-Filter optimale Schätzung für x nachdem eine Messung z beobachtet wurde ergibt sich durch Gleichung 2.5.

$$\hat{x}_{neu} = \hat{x} + \Sigma H^T (H\Sigma H^T + R)^{-1}(z - H\hat{x}) \tag{2.5}$$

Gleichzeitig wird die Kovarianz nach Gleichung 2.6 ermittelt.

$$\Sigma_{neu} = \Sigma - \Sigma H^T (H\Sigma H^T + R)^{-1} H\Sigma \tag{2.6}$$

2.2.2 Der erweiterte Kalman Filter (EKF)

Ein Problem des Kalman-Filters ist es, dass nur lineare Prozesse betrachtet werden können. Dies ist insbesondere in der Robotik ein Problem, da Propagationen und Messungen häufig nicht linear sind. Aus diesem Grund verwendet man den erweiterten Kalman-Filter (EKF) zur Schätzung von nicht-linearen Problemen mit dem Kalman-Filter. Die grundsätzliche Idee dabei ist es die nichtlinearen Funktionen durch eine Taylor-Approximation erster Ordnung zu linearisieren [24]. Dabei stellen sich die Propagations- und Messgleichung für den EKF als nicht-lineare Gleichungen dar.

$$x = g(x, u) + \varepsilon \tag{2.7}$$
$$z = h(x) + \omega \tag{2.8}$$

Um auf diese den Kalman-Filter anwenden zu können werden daher die Matrizen A, B und H durch die Jacobi-Matrizen $\frac{\partial g(x,u)}{\partial x}$, $\frac{\partial g(x,u)}{\partial u}$ und $\frac{\partial h(x)}{\partial x}$ an der Stelle von $\hat{x}$ ersetzt, wenn dies erforderlich ist. Die Funktionen $g(x, u)$ und $h(x)$ können weiterhin verwendet werden, um zum Beispiel beim Mess-Schritt den Term $z - h(\hat{x})$ zu berechnen.

Der erweiterte Kalman-Filter liefert damit immer noch zuverlässige Messungen, auch für nicht-lineare Größen. Verläuft die zu approximierende Funktion allerdings allzu nichtlinear, so kann der Filter divergieren und er verliert damit seine Optimalität. Verschiedene Arbeiten [16, 5] haben aber gezeigt, dass trotz dieses theoretischen Problems die Anwendung des EKF in der Praxis möglich ist.

2.3 Hough-Transformation

Die Hough-Transformation dient dazu Linien oder im allgemeinen Fall geometrische Strukturen in Punktmengen zu erkennen. Die Idee besteht darin die Punktmenge in den Raum der Linien zu transformieren, wobei jeder Punkt für die Menge aller möglichen Linien steht, die durch diesen Punkt verlaufen [10]. Bei der Transformation wird für jeden Punkt die Menge dieser Linien betrachtet und jeweils ein Zähler im sogenannten Akkumulator erhöht. Die parametrisierte Linie mit dem größten Zähler ist dann die Gerade in der Punktmenge, die durch die meisten Punkte verläuft und wird somit als beste Schätzung erkannt. Von den verschieden Möglichkeiten Geraden zu parametrisieren hat sich die Darstellung durch Winkel θ der Geraden zur x-Achse und ρ als Abstand zum Ursprung (Gleichung 2.9) als geeignet gezeigt.

$$\rho = x \cdot \cos\theta + y \cdot \sin\theta \tag{2.9}$$

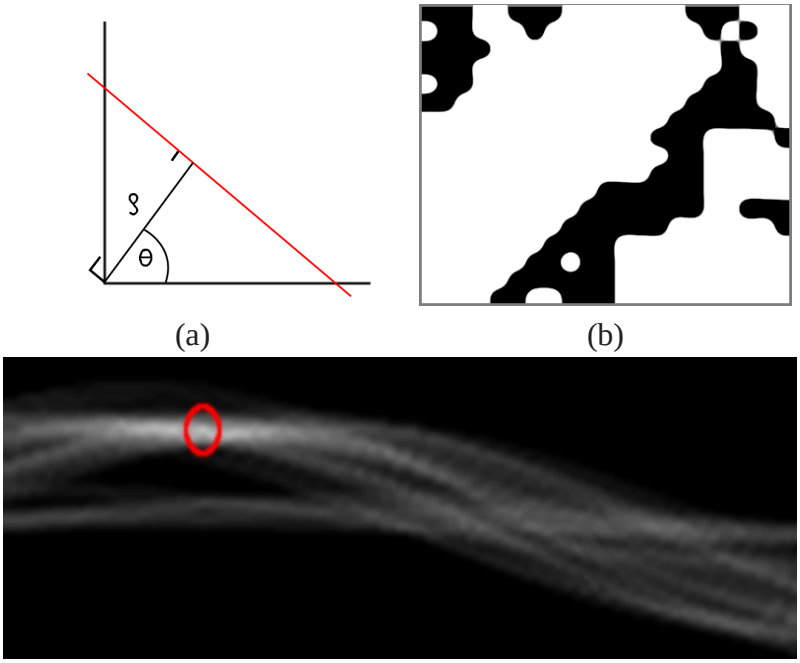
Bei der Transformation wird dann für jeden Punkt (x, y) der Punktmenge und für jeden Geradenwinkel θ der Abstand ρ ermittelt und an der Stelle (θ, ρ) im Hough-transformierten Raum ein Zähler erhöht. Das Finden der besten Gerade geschieht dann einfach dadurch, dass man die Gerade (θ^*, ρ^*) im Hough-transformierten Raum mit dem größten Zähler ermittelt.

2.3.1 Mehrfache Hough-Transformation

Es ist eventuell wünschenswert nicht nur eine Linie zu erkennen, sondern mehrere Linien. Die einfache Anfrage nach dem zweitbesten oder drittbesten Punkt im Hough-transformierten Raum liefert aber meist keine guten Ergebnisse, da diese meist Geraden liefert, die der besten Gerade sehr ähnlich sind, weil zwei Geraden mit fast gleichen Parametern viele Punkte gemeinsam haben und somit auch einen hohen Zählerstand erreichen. Eine solche Gerade möchte man in der Regel nicht erkennen, da diese keine neuen Informationen liefert. Eine einfache praktikable Lösung ist es deshalb nach der Ermittlung der besten Linie eine weitere Hough-Transformation durchzuführen, wobei bei dieser die Punkte, die auf der schon erkannten Gerade liegen, nicht mehr transformiert werden. Dadurch findet ein weiterer Durchlauf nur Geraden, die sich von den schon erkannten unterscheiden.

2.4 Markov Random Fields

Die Klassifikation von komplexen Daten, ist immer dann schwierig, wenn sich die eigentliche Klasse nicht aus einem Datenpunkt erschliessen lässt, sondern dessen korrekte Klassifikation sich erst aus der Verbindung von mehreren Daten erschließt,



(a) (b)

(c)

Abbildung 2.2. Visualisierung der Hough Transformation

Visualisierung der Hough Transformation: (a) Parametrisierung einer Geraden mittels θ und ρ (b) Eingabepunktmenge für die Hough-Transformation. (c) Die Hough-Transformation der Punktmenge. Der Akkumulator zeigt die beste Linie als hellsten Punkt im Hough-transformierten Raum, der hier durch einen roten Kreis markiert ist.

wie zum Beispiel bei einer dreidimensionalen Weltdarstellung. Herkömmliche Methoden, die Merkmale nur auf einzelnen Daten bestimmen und diese klassifizieren, haben bei solchen Problemen oftmals Schwierigkeiten. Markov Random Fields modellieren explizit auch den Zusammenhang zwischen Komponenten und sind somit ein vielversprechender Ansatz, um untereinander verknüpfte Daten zu klassifizieren.

Die zu klassifizierenden Objekte werden dazu als eine Menge von diskreten Variablen $\mathrm{Y} = \{Y_1, \ldots, Y_N\}$ symbolisiert, wobei jede Variable Y_i den K möglichen Klassen zugeordnet werden kann, so dass $Y_i \in \{1, \ldots, K\}$. Über diese Variablen wird ein ungerichteter Graph $\mathcal{G} = (\mathrm{Y}, \mathcal{E})$ definiert. Eine Variablenbelegung y wird dabei durch die Menge $\{y_i^k\}$ von Indikatorfunktionen repräsentiert, wobei $y_i^k = I(y_i = k)$.

In dieser Arbeit wird der Ansatz von *Anguelov et al.* [2] verfolgt, der *paarweise* Markov Random Fields benutzt. Dabei ist jedem Knoten des Graphen eine Potentialfunktion $\phi(y_i)$ zugeordnet und jeder Kante $(i,j) \in \mathcal{E}$ eine Potentialfunktion $\phi(y_i, y_j)$. Ein paarweises MRF definiert dann eine Wahrscheinlichkeitsverteilung durch:

$$P_\phi(\mathrm{y}) = \frac{1}{Z} \prod_{i=1}^{N} \phi_i(y_i) \prod_{(ij)\in\mathcal{E}} \phi_{ij}(y_i, y_j) \tag{2.10}$$

Dabei ist Z eine Normalisierungskonstante, die gegeben ist durch:
$Z = \sum_{\mathrm{y}'} \prod_{i=1}^{N} \phi_i(y_i') \prod_{(ij)\in\mathcal{E}} \phi_{ij}(y_i', y_j')$.

Um die Klassifikation benachbarter Objekte zu einer Klasse zu belohnen, werden *assoziative Markov Netzwerke* [27] benutzt. Dazu wird gefordert, dass die Potentialfunktion für Kanten definiert ist durch Gleichung 2.11 mit $\lambda_{ij}^k \geq 1$.

$$\phi_{ij}(k,l) = \begin{cases} \lambda_{ij}^k & \text{wenn } k = l \\ 1 & \text{sonst} \end{cases} \tag{2.11}$$

Im weiteren soll zur Darstellung mittels der log-Wahrscheinlichkeit $\log P_\phi(y)$ übergegangen werden. Die Potentialfunktion können dann über Merkmalsvektoren $x_i \in \mathbb{R}^{d_n}$ für Knoten mit d_n Merkmalen und $x_{ij} \in \mathbb{R}^{d_e}$ für Kanten mit d_e Merkmalen realisiert werden.

$$\log \phi_i(k) = \mathrm{w}_n^k \cdot \mathrm{x}_i \tag{2.12}$$

$$\log \phi_{ij}(k,k) = \mathrm{w}_e^k \cdot \mathrm{x}_{ij} \tag{2.13}$$

Dabei sind w_n^k und w_e^k Zeilenvektoren der Dimension d_n, beziehungsweise d_e, die als Gewichtsvektoren der Klasse k für die Merkmale verstanden werden. Des weiteren wird gefordert, dass $w_e^k \geq 0$ und $x_{ij} \geq 0$, um Gleichung 2.11 sicherzustellen.

Die Klassifikation in Markov Random Fields wird durch die Lösung des *maximum a-posteriori (MAP)* Inferenz Problems bewerkstelligt. Das heißt, gesucht ist:

$\arg\max_{\mathrm{y}} P_\phi(\mathrm{y})$. Mit den Gleichungen 2.12 und 2.13 ergibt sich dann die zu lösende Gleichung 2.14.

$$\arg\max_{\mathrm{y}} \sum_{i=1}^{N}\sum_{k=1}^{K}(\mathrm{w}_n^k \cdot \mathrm{x}_i)y_i^k + \sum_{(ij)\in\mathcal{E}}\sum_{k=1}^{K}(\mathrm{w}_e^k \cdot \mathrm{x}_{ij})y_i^k y_j^k. \tag{2.14}$$

Diese lässt sich als lineares Optimierungsproblem formulieren [2], das zum Beispiel mit dem *Simplex*-Verfahren gelöst werden kann.

$$\begin{aligned} &\max \sum_{i=1}^{N}\sum_{k=1}^{K}(\mathrm{w}_n^k \cdot \mathrm{x}_i)y_i^k + \sum_{(ij)\in\mathcal{E}}\sum_{k=1}^{K}(\mathrm{w}_e^k \cdot \mathrm{x}_{ij})y_{ij}^k \\ &\text{s.t. } y_i^k \geq 0, \quad \forall i,k; \qquad \sum_k y_i^k = 1, \quad \forall i; \\ &\quad y_{ij}^k \leq y_i^k, \quad y_{ij}^k \leq y_j^k, \quad \forall ij \in \mathcal{E}, \forall k \end{aligned} \tag{2.15}$$

Dazu ist es nötig den quadratischen Term $y_i^k y_j^k$ durch die Hilfsvariablen y_{ij}^k mit den zusätzlichen Bedingungen $y_{ij}^k \leq y_i^k$ und $y_{ij}^k \leq y_j^k$ zu ersetzen.

Es ist weiterhin notwendig die Knoten- und Kantengewichte aus von Hand klassifizierten Daten zu lernen. Dazu werden *maximum margin Markov Netzwerke* benutzt, die bei *Taskar et al.* [27] näher beschrieben werden. Eine Implementierung dieses Lernalgorithmus, die somit benutzt werden konnte, war bereits vorhanden [15].

2.5 A*-Pfadplanung

Der A*-Algorithmus ist ein Suchalgorithmus, der unter anderem in der Robotik zur Pfadplanung eingesetzt wird. Dabei wird in einem ungerichtetem Graph $G = (V, E)$, dessen Knoten den Zellen einer Karte entsprechen, ein Pfad von einem Start- zu einem Zielknoten gesucht. Mit dem Graphen ist eine Kostenfunktion $c : E \to \mathbb{R}_+$ verknüpft, die angibt, wie teuer es ist, von einem Knoten zu einem anderen zu gelangen. Des weiteren gibt es einen Startknoten $s \in V$, der in der Regel der aktuellen Roboterposition entspricht und einen Zielknoten $g \in V$, zu dem der kürzeste Weg berechnet werden soll. Um die Suche zu beschleunigen wird eine Heuristik $h(n)$ benutzt, die zu einem Knoten n eine Abschätzung der Entfernung zum Ziel angibt. Gilt für alle Knoten $n \in V$, dass $h(n) \leq h^*(n)$, wobei h^* jeweils die Länge des kürzesten Pfades zum Ziel ist, so spricht man von einer zulässigen Heuristik. Die Ausgabe des Algorithmus ist ein Pfad im Graph G, der, wenn h zulässig ist, minimale Kosten hat [22]. Als Datenstrukturen kommen dabei eine Warteschlange („Queue“) zur Speicherung der zu expandierenden Knoten und eine Menge zur

Algorithmus 1 : A*-Suche

Eingabe : Graph $G = (V, E)$, Kostenfunktion $c : E \to \mathbb{R}_+$, Startknoten $s \in V$, Zielknoten $g \in V$

```
Warteschlange q;
Menge closedList;
begin
    q ← Insert(h(s), s)
    while (n = q ← PopFront()) ≠ g do
        closedList ← Insert(n)
        Expansion = ExpandNode(n)
        for e ∈ Expansion do
            if e ∉ closedList then
                g(e) = g(n) + c(n, e)
                parent(e) ← n
                q ← Insert(h(e) + g(e), e)
        if q ← empty then
            return Failure
end
```

Speicherung der schon expandierten Knoten („closed list“) zum Einsatz. Der Algorithmus 1 beschreibt dabei den grundsätzlichen Ablauf, der darin besteht, dass der jeweils beste Knoten aus der Queue entfernt und expandiert wird, bis der Zielknoten erreicht, oder die Queue leer ist. Die Funktion $ExpandNode$ liefert dabei alle traversierbaren Nachbarknoten eines Knotens und die $closedList$, die die bereits expandierten Knoten enthält, dient dazu, die Suche in Zykeln zu vermeiden, die in einem Graph vorkommen können, so dass jeder Knoten nur einmal expandiert wird. Als Bewertungsfunktion für einen Knoten benutzt der A* dabei die Funktion $f(n) = g(n)+h(n)$, wobei $g(n)$ die aktuellen Kosten des Knotens sind und $h(n)$ die heuristische Abschätzung ist. Dies gewährleistet bei einer zulässigen Heuristik die Optimalität des gefundenen Pfades. Der Pfad ergibt sich dann letzen Endes durch rekursives Zurückverfolgen der $parent$-Eigenschaft vom Zielknoten ausgehend.

Kapitel 3

Weltmodellierung mit Höhenkarten

Die Qualität und Aussagekraft des Weltmodells haben einen direkten Einfluss darauf, ob der Roboter überhaupt eine Chance hat, sich in der Umgebung zurecht zu finden. Klassische Ansätze, wie zum Beispiel Occupancy-Grids [7] gehen von einer zweidimensionalen Welt aus, in der es nur freie Flächen oder belegte Flächen gibt. Diese Information reicht für das Überwinden von Hindernissen bei weitem nicht aus. Die Speicherung der kompletten geometrischen Rohdaten, wie zum Beispiel Punktwolken, wie sie von 3D-Laserscannern erzeugt werden, gibt die Welt zwar genau wieder, erfordert allerdings sehr hohen Speicherbedarf. Deshalb sollen Höhenkarten verwendet werden, die als $2\frac{1}{2}$-dimensionale Weltrepräsentation ein gutes Mittelmaß darstellen. Höhenkarten sind zellenbasierte Karten, die in jeder Zelle die Höhe dieser Zelle speichern und somit ein gutes Abbild der Umgebung ermöglichen.

3.1 Aufbau von Höhenkarten

Um Höhenkarten während der Fahrt aufbauen zu können, wird ein Laserscanner nach unten geschwenkt, der die Umgebung wahrnimmt (siehe Abbildung 1.1(b)). Dessen Daten werden zusammen mit Positionsdaten, die durch einen 2D-SLAM-Algorithmus von einem horizontal montierten Laserscanner erzeugt werden, und den Winkelinformationen eines Lagesensors zu einer Höhenkarte integriert. Dies geschieht grundsätzlich in zwei Schritten:

Zuerst werden dazu die Daten des 2D-Laserscanners, die in lokalen Polarkoordinaten vorliegen, in kartesische Koordinaten transformiert, so dass man dreidimensionale Punkte in Weltkoordinaten erhält. Im zweiten Schritt werden diese Punkte mit einem Kalman-Filter zu einer Höhenkarte integriert.

3.1.1 Koordinatentransformationen

Um die lokalen Polarkoordinaten ins (xz)-Roboterkoordinatensystem (siehe Abschnitt 2.1) und anschließend in kartesische Weltkoordinaten zu konvertieren, wer-

den der Reihe nach folgende Transformationen vorgenommen, wobei die Lage der verschiedenen Koordinatensysteme in Abbildung 3.1 schematisch dargestellt ist.

1. Umwandlung von Polarkoordinaten bezüglich des Laserscanner Referenzsystems in kartesische (xz)-Koordinaten.

2. Transformation vom Laserscannerkoordinatensystem ins Koordinatensystem der Kippvorrichtung.

3. Transformation vom Koordinatensystem der Kippvorrichtung ins Roboterkoordinatensystem.

4. Transformation vom Roboterkoordinatensystem ins Weltkoordinatensystem.

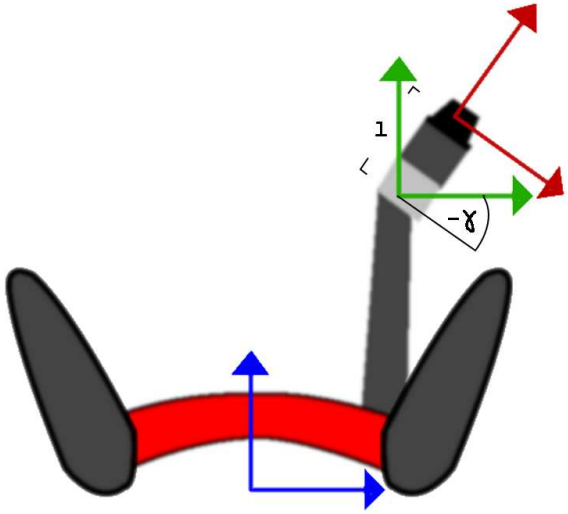


Abbildung 3.1. Koordinatensysteme des Roboters: Laserscannerkoordinatensystem (rot), Koordinatensystem der Kippvorrichtung (grün) und Roboterkoordinatensystem (blau).

Zu 1: Seien die Polarkoordinaten gegeben als (d_i, α_i), wobei d_i die gemessene Entfernung zum Scannermittelpunkt und α_i den Winkel des Strahls bezeichnet. Dann ergeben sich die kartesischen Koordinaten $\mathbf{x}_i^1$ durch Gleichung 3.1.

$$\mathbf{x}_i^1 = (d_i \cdot \sin \alpha_i, 0, d_i \cdot \cos \alpha_i)^T \tag{3.1}$$

Zu 2: In Gleichung 3.2 werden die Punkte $\mathbf{x}_i^1$ um die Kippachse mit Hebellänge l um den Winkel γ gedreht.

$$\mathbf{x}_i^2 = R_x(\gamma) \cdot \left(\mathbf{x}_i^1 + (0, l, 0)^T\right) \tag{3.2}$$

Zu 3: Da die Kippvorrichtung sich nicht im Mittelpunkt des Roboters befindet, sondern um $(s_x, s_y, s_z)^T$ verschoben ist, werden die Punkte vom Koordinatensystem der Kippvorrichtung durch Gleichung 3.3 ins Roboterkoordinatensystem transformiert.

$$\mathbf{x}_i^3 = \mathbf{x}_i^2 + (s_x, s_y, s_z)^T \tag{3.3}$$

Zu 4: Der Roboter befindet sich an der Position $(x, z)^T$ in der Welt und schaut in Richtung θ. Da der Roboter sich in einer dreidimensionalen Welt befindet, ist außerdem die Roboterhöhe h in der Welt, sowie dessen Roll-(ζ) und Nickwinkel(η) relevant. Gleichung 3.4 transformiert die Punkte x_i vom Roboterkoordinatensystem mit der Roboterpose $(x, h, z, \theta, \eta, \zeta)^T$ in Weltkoordinaten.

$$\mathbf{x}_i^4 = (x, h, z) + R_z(\zeta) \cdot R_y(\eta) \cdot R_x(\theta) \cdot \mathbf{x}_i^3 \tag{3.4}$$

3.1.2 Datenintegration

Die nach den Transformationen erhaltenen 3D-Weltpunkte werden nun zellenweise integriert. Dazu wird zuerst festgestellt, welcher Punkt zu welcher Zelle korrespondiert und dieser dann mit dem Kalman-Filter als Höhenabschätzung für diese Zelle verschmolzen [21]. Dabei kann es allerdings vorkommen, dass eine Messung z_h unterhalb der wirklichen Höhe des Objektes liegt. Dies geschieht insbesondere dann, wenn der Roboter ein vertikales Objekt, wie zum Beispiel eine Wand, sieht (siehe Abbildung 3.2). Um diese Problematik zu berücksichtigen, wird die Höhe nur dann durch den Kalman-Filter integriert, wenn die aktuelle Messung innerhalb der Mahalanobis-Distanz von $d_M(z_h) \leq c$ mit $c = 1$ liegt. Ansonsten wird bei einer höheren Messung die Höhe der Zelle mit einer Messvarianz von 0 in den Kalman-Filter integriert. Dies hat den Effekt, dass die Höhe der Zelle auf die der Messung gesetzt wird, da diese jetzt eine bessere Abschätzung für die Objekthöhe liefert. Umgekehrt wird bei einer niedrigeren Messung diese nur schwach (das heißt mit einer um eine Größenordnung erhöhten Varianz) integriert, da eine solche Messung eine ungenauere Abschätzung für die wirkliche Zellenhöhe liefert. Es ergibt sich Gleichung 3.5 für die Integration einer Messhöhe z_h zur aktuellen Höhenabschätzung $\widehat{h}_t$.

$$\widehat{h}_t = \begin{cases} z_h & \text{wenn } d_M(z_h) > c \text{ und } z_h > \widehat{h}_{t-1} \\ \frac{\sigma_h^2 \cdot z_h + 10 \cdot \sigma_z^2 \cdot \widehat{h}_{t-1}}{\sigma_h^2 + 10 \cdot \sigma_z^2} & \text{wenn } d_M(z_h) > c \text{ und } \widehat{h}_{t-1} > z_h \\ \frac{\sigma_h^2 \cdot z_h + \sigma_z^2 \cdot \widehat{h}_{t-1}}{\sigma_h^2 + \sigma_z^2} & \text{sonst} \end{cases} \tag{3.5}$$

Dabei ist $d_M(z) = \sqrt{\frac{\left(\widehat{h}_{t-1} - z\right)^2}{\sigma_h^2}}$.

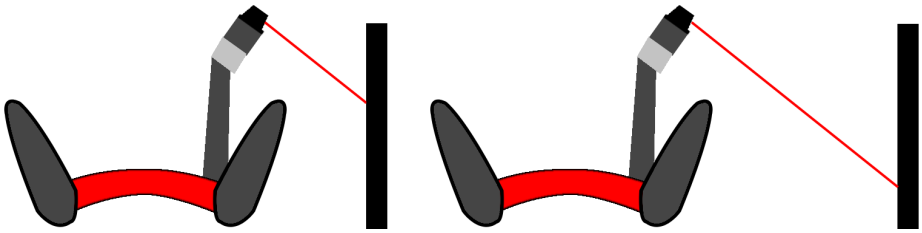


Abbildung 3.2. Datenintegration bei Annäherung an eine Wand: Je näher der Roboter eine Wand observiert, desto höher ist die jeweilige Messung, obwohl die Wand in beiden Fällen die gleiche Höhe besitzt.

3.2 Bestimmen der Transformations-Parameter

Die bei den Koordinatentransformationen verwandten Parameter müssen zur Durchführung der Transformationen bekannt sein. Es soll jetzt gezeigt werden, wie diese Parameter bestimmt werden können.

Statische Parameter Die Kipphebellänge l, sowie die Lage der Kippvorrichtung relativ zum Robotermittelpunkt $(s_x, s_y, s_z)^T$ sind durch die Mechanik vorgegeben und wurden durch einfaches Nachmessen ermittelt.

Dynamische Parameter Der Kippwinkel γ des Laserscanners ist einstellbar und wird auf $-35°$ initialisiert. Änderungen am Kippwinkel werden von der Mikrocontrollersoftware mitgeteilt. Die Roll- und Nickwinkel ζ und η werden direkt von dem angebrachten Lagesensor entnommen.

3.2.1 Ermitteln der Roboterpose

Um eine dreidimensionale Höhenkarte zu bauen, ist es notwenig eine Pose in allen sechs Freiheitsgraden zu bestimmen. Dies wird durch die Verwendung des Lagesensors, der die Roll- und Nickwinkel bereitstellt erleichtert. Die Roboterpose $(x, z, h)^T$, sowie die Roboterrichtung θ können allerdings nicht direkt bestimmt werden. Deshalb wird hier ein vorhandener 2D-SLAM-Algorithmus [14, 8] angewandt, der eine 2D-Pose relativ zu der Ebene liefert, in der der Roboter sich gerade befindet. Da die erforderlichen (xz)-Koordinaten als Weltkoordinaten in der (xz)-Ebene liegen müssen, werden die Koordinaten des 2D-SLAM-Algorithmus auf die Ebene projiziert, falls der Roboter sich auf einer Schrägen bewegt. Zusätzlich muss die Roboterhöhe h bestimmt werden.

Zum Schätzen der dreidimensionalen Roboterpose $(x, z, h)^T$ wird, wie schon bei der Datenintegration, ein Kalman-Filter angewandt, der in diesem Fall als EKF (siehe Abschnitt 2.2.2) implementiert ist. Dieser bekommt als Eingaben für den

Vorhersage-Schritt den Abstand d der 2D-Pose des SLAM-Algorithmus zur vorherigen 2D-Pose, die Richtung des Roboters θ aus dem SLAM-Algorithmus und vom Lagesensor den Nickwinkel η. Bewegt sich der Roboter vorwärts und befindet sich dabei auf einer Schräge, so ändert sich die Roboterhöhe entlang des Nickwinkels, die sich so ermitteln lässt.

Die nicht-lineare Funktion $f(l,u)$ für den Vorhersageschritt bekommt als Zustandvektor l die aktuelle Pose des Kalman-Filters $(\widehat{x},\widehat{z},\widehat{h})^T$ und als Eingangsvektor u den Parametervektor $(\eta,\theta,d)^T$. Gleichung 3.6 zeigt, wie sich daraus im hier verwendeten (xz)-Koordinatensystem (siehe 2.1) eine Abschätzung $(x,z,h)^T$ im Vorhersage-Schritt berechnen lässt. Zur Berechnung der neuen Kovarianzmatrix Σ_{l_t} muss sowohl die alte Kovarianzmatrix $\Sigma_{l_{t-1}}$ als auch die Kovarianzmatrix Σ_u, die den Fehler des Eingabevektors wiedergibt, mit $f(l,u)$ transformiert werden (Gleichung 3.7). Da $f(l,u)$ allerdings keine lineare Funktion ist, werden hier die Jacobi-Matrizen $\nabla F_l = \frac{\partial f(l,u)}{\partial l}$ zur Transformation der alten Kovarianzmatrix $\Sigma_{l_{t-1}}$ und $\nabla F_u = \frac{\partial f(l,u)}{\partial u}$ für die Eingangskovarianzmatrix benutzt.

$$\begin{pmatrix} \widehat{x}_t \\ \widehat{z}_t \\ \widehat{h}_t \end{pmatrix} = \begin{pmatrix} \widehat{x}_{t-1} - d \cdot \sin\theta \cdot \cos\eta \\ \widehat{z}_{t-1} - d \cdot \cos\theta \cdot \cos\eta \\ \widehat{h}_{t-1} + d \cdot \sin\eta \end{pmatrix} \tag{3.6}$$

$$\Sigma_{l_t} = \nabla F_l \Sigma_{l_{t-1}} \nabla F_l^T + \nabla F_u \Sigma_u \nabla F_u^T \tag{3.7}$$

Dabei sind die verwendeten Matrizen durch Gleichung 3.8 bis 3.11 definiert.

$$\nabla F_l = \frac{\partial f}{\partial (x,z,h)^T} \Big|_{(\widehat{x}_{t-1},\widehat{z}_{t-1},\widehat{h}_{t-1})^T} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.8}$$

$$\begin{aligned} \nabla F_u &= \frac{\partial f}{\partial (\eta,\theta,d)^T} \Big|_{(\widehat{x}_{t-1},\widehat{z}_{t-1},\widehat{h}_{t-1})^T} \\ &= \begin{pmatrix} d\sin\theta\sin\eta & -d\cos\theta\cos\eta & -\sin\theta\cos\eta \\ d\cos\theta\sin\eta & d\sin\theta\cos\eta & -\cos\theta\cos\eta \\ d\cos\eta & 0 & \sin\eta \end{pmatrix} \end{aligned} \tag{3.9}$$

$$\Sigma_{l_{t-1}} = \begin{pmatrix} \sigma_x^2 & \sigma_{xz}^2 & \sigma_{xh}^2 \\ \sigma_{xz}^2 & \sigma_z^2 & \sigma_{zh}^2 \\ \sigma_{xh}^2 & \sigma_{zh}^2 & \sigma_h^2 \end{pmatrix} \tag{3.10}$$

$$\Sigma_u = \begin{pmatrix} \sigma_\eta^2 & 0 & 0 \\ 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & \sigma_d^2 \end{pmatrix} \tag{3.11}$$

Um ein Abdriften der Höhe zu verhindern, wird im Mess-Schritt des Kalman-Filters die Höhe der Zelle h_K, auf der der Roboter sich in der Höhenkarte befindet,

als Rückmeldung für dessen Höhe benutzt. Dabei gilt für die Mess-Funktion h_m, dass $h_m\left((x,z,h)^T\right) = h$, womit die Jacobi-Matrix $H = \frac{\partial h_m}{\partial l}$ durch $(0,0,1)$ gegeben ist. Die dementsprechende Messvarianz $R = \sigma^2_{h_K}$ ist durch die Varianz der Zelle in der Höhenkarte gegeben, die für die Höhenmessung benutzt wird. Einsetzen in den Kalman-Filter führt zu Gleichung 3.12 für die Berechnung von $\widehat{h}_t$ aufgrund von $\widehat{h}_{t-1}$ und σ^2_h, sowie der Messung h_K mit $\sigma^2_{h_K}$.

$$\widehat{h}_t = \frac{\sigma^2_h h_K + \sigma^2_{h_K}\widehat{h}_{t-1}}{\sigma^2_{h_K} + \sigma^2_h} \tag{3.12}$$

3.3 Varianz-Propagation

Beim Kalman-Filter wechseln sich in der Regel immer zwei Schritte ab: Vorhersage und Messung. Bisher wurde bezüglich des Kalman-Filters, der die Höhenmessungen in die Karte integriert, nur auf den Mess-Schritt eingegangen. Der Vorhersage-Schritt ist dabei grundsätzlich relativ einfach, da sich durch die Roboterbewegung die Karte nicht ändert, das heißt beim Vorhersage-Schritt werden die Zellhöhen nicht verändert. Allerdings nimmt die Unsicherheit über die Höhe einer Zelle mit der Zeit zu, da die Informationen veraltet sein können. Dies ist insbesondere dadurch relevant, dass die Integration neuer Daten auf der Varianz einer Zelle beruht. Erreicht der Roboter, nachdem er eine gewisse Strecke gefahren ist, eine Position wieder, die er vorher schon gesehen hat, so ist die Genauigkeit mit der die Roboterposition bestimmt werden kann und damit auch die Zuordnung der aktuell zu integrierenden Daten zu der passenden Zelle deutlich geringer als bei der ersten Sichtung. Da die Karte aus der Perspektive des Roboters gebaut wird, sind die aktuellen Informationen stärker zu bewerten als ältere. Um die Ungenauigkeit der Datenzuordnung in Abhängigkeit davon, wie lange der Roboter diese nicht gesehen hat, zu modellieren, wird, wenn der Roboter sich bewegt, die Varianz erhöht.

Gleichung 3.13 gibt dabei den Vorhersage-Schritt an. Es ist zu beachten, dass die Höhe nicht geändert wird und deshalb auch die Varianztransformation der Einheitsmatrix entspricht. Es wird lediglich die Unsicherheit der Abschätzung Q, die als lineares Modell angenommen wird, addiert (Gleichung 3.14).

$$\widehat{h}_t = \widehat{h}_{t-1} \tag{3.13}$$

$$\sigma^2_h = 1 \cdot \sigma^2_h + \underbrace{(\sigma^2_d \cdot d + \sigma^2_a \cdot a)}_{Q} \tag{3.14}$$

Dabei ist d die gefahrene Strecke seit dem letzten Schritt und a der gefahrene Winkel, also die Drehung des Roboters. Des weiteren sind $\sigma^2_d[mm^2/mm]$ und $\sigma^2_a[mm^2/^\circ]$

Gewichtungsfaktoren, die wiedergeben, wie stark sich die Varianz mit der gefahrenen Strecke, beziehungsweise dem gefahrenen Winkel erhöht. Diese Parameter wurden experimentell bestimmt.

Würde man diese Varianzerhöhungen kontinuierlich vornehmen, so hätte das allerdings den Nachteil, dass bei jeder Roboterbewegung (beziehungsweise jedem Erhalten einer neuen Lokalisierungspose) sämtliche Zellen der Karte betrachtet werden müssten. Dies kann dadurch umgangen werden, dass man den Vorhersage-Schritt bis zur nächsten Messung verzögert. Bezüglich der Karte ist dies kein Problem, da sich die Höhenabschätzung ja nicht ändert. Will man allerdings die Vorhersage-Schritte bis zur nächsten Messung herauszögern, so müssen deren Parameter bekannt sein. Das heißt konkret, dass man für jeden beliebigen Zeitpunkt feststellen können muss, welche Strecke und welchen Winkel der Roboter seitdem gefahren ist. Das Nachhalten dieser Größe für jeden Zeitpunkt und jede Zelle, würde das Problem somit nur verlagern. Es ist allerdings möglich, diese Größe in konstanter Zeit für alle Zellen nachzuhalten und zu berechnen, wenn man zur Integraldarstellung der Distanz übergeht, analog dazu, wie der Algorithmus von Viola-Jones [32] beim Erstellen des Integral-Image vorgeht. Dazu bezeichnet $d(t,k)$ die Distanz, die der Roboter zwischen den Zeitpunkten t und k zurückgelegt hat. Gleichung 3.15 bis 3.17 stellen die Zusammenhänge dar.

$$d(t,k) := \int_t^k d(t', t'-1)\,\mathrm{d}t' \tag{3.15}$$

$$d(t,k) = \int_0^k d(t', t'-1)\,\mathrm{d}t' - \int_0^t d(t', t'-1)\,\mathrm{d}t' \tag{3.16}$$

$$d(0,t) = \int_0^{t-1} d(t', t'-1)\,\mathrm{d}t' + d(t-1,t) \tag{3.17}$$

Dabei definiert Gleichung 3.15 $d(t,k)$, also die gesuchte Größe. Diese wird allerdings nicht für alle t,k gespeichert, sondern es wird nur der Wert $d(0,t)$ nachgehalten. Gleichung 3.17 zeigt, wie bei einer zu integrierenden Strecke der Wert für $d(0,t)$ aus den schon gespeicherten Werten direkt errechnet werden kann. Diese Berechnung geschieht allerdings nur einmal pro neuer Lokalisierungspose und ist somit nicht von der Größe der Karte abhängig. Mit $d(0,t)$ für alle t lässt sich dann für beliebige t,k der gesuchte Wert $d(t,k)$ aus Gleichung 3.16 berechnen.

In der Implementierung wird dabei $d(0,t)$ mit dem Abstand von einer Sekunde diskretisiert, was die einfache Speicherung der Daten ermöglicht. Dabei eventuell auftretende Diskretisierungsfehler liegen damit unterhalb einer Sekunde und sind bei der Fahrgeschwindigkeit der Roboters vernachlässigbar. Für das Nachhalten des Winkels wird die gleiche Größe analog mit den jeweiligen Winkelunterschieden zwischen zwei Posen berechnet.

3.4 Karten-Filterung

Da das Erstellen von Höhenkarten auf der Integration von Messdaten beruht, können dort Fehler auftauchen, die die Qualität der Karte verringern. Dies ist zum einen Messrauschen, das sich durch Dellen in glatten Flächen, wie zum Beispiel Böden zeigt und durch das Schwanken des Roboters während der Fahrt entsteht. Zum anderen treten dadurch, dass einige Zellen vom Laserscanner nicht gesehen wurden, Löcher in der Karte auf. Solche Ungenauigkeiten im Weltmodell, machen eine korrekte Klassifikation der Karte unmöglich und stellen deshalb auch für nachfolgende Algorithmen ein großes Problem dar.

Diese Probleme sollen durch eine geeignete Filterung der Karte vermindert werden. Dem zugrunde liegt die Annahme, dass eine Karte sich in den meisten Fällen stetig verhält, so dass man die Nachbarzellen in die Höhenabschätzung einer Zelle mit einbeziehen kann. Parameter für die neue Zellhöhe sind die Höhen der umgebenden Zellen, die mit einem entsprechenden Gewicht in die Berechnung der neuen Zellhöhe eingehen. Dazu werden für Höhenkarten zwei Gewichtsfaktoren berechnet, die sich einerseits aus der Nähe zu der Zelle und andererseits aus der Varianz σ^2, beziehungsweise der Sicherheit $\frac{1}{\sigma^2}$ berechnen. Der erste Gewichtsfaktor w_1, der den Abstand um eine Zelle widergibt, wird dabei nach der Verteilung in Tabelle 3.1 um die aktuelle mittlere Zelle vorgenommen. Weiterhin wird ein zweiter Gewichtsfaktor

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

Tabelle 3.1. Gewichtsfaktoren für umliegende Zellen.

w_2 aus der Varianz jeder Zelle erzeugt, so dass Zellen mit einer geringeren Varianz stärker bewertet werden. Dies führt insbesondere dazu, dass Löcher in der Karte, die von nicht initialisierten Zellen stammen, vollständig geschlossen werden, da deren Varianz sehr groß ist. Dabei ist $w_2 = \frac{1}{\sigma_i^2}$ wobei σ_i^2 die Varianz der jeweiligen Zelle bezeichnet. Jede Zellhöhe der umliegenden Zellen wird mit $w_1 \cdot w_2$ gewichtet und die gefilterte Höhe der Zelle berechnet sich dann abhängig vom angewandten Filterverfahren. Bekannte Verfahren sind der gewichtete Mittelwert und der gewichtete Median [36]. Seien dazu die Gewichte w_i und die Höhen h_i der n Nachbarzellen gegeben, so berechnet sich der gewichtete Mittelwert h_{mean} nach Gleichung 3.18.

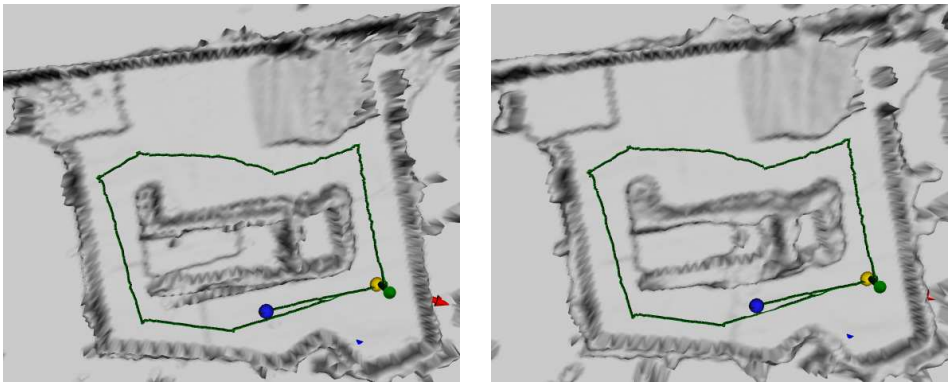
$$h_{mean} = \frac{\sum_{i=1}^{n} w_i \cdot h_i}{\sum_{i=1}^{n} w_i} \tag{3.18}$$

Zur Berechnung des gewichteten Median, werden die Höhen h_i und die Gewichte w_i nach den Höhen sortiert. Der gewichtete Median h_{median} ist die Höhe h_j mit dem

größten j, so dass gilt:

$$\sum_{i=1}^{j} w_i \leq 0.5 \cdot \sum_{i=1}^{n} w_i$$

Im Laufe der Diplomarbeit wurden sowohl der gewichtete Mittelwert als auch der gewichtete Medianfilter implementiert und schließlich der Medianfilter benutzt, da dieser bessere Eigenschaften in der Unterdrückung von Löchern bietet als ein Mittelwertfilter. Ein Beispiel der Anwendung des Median-Filters ist in Abbildung 3.3 zu sehen.



(a) (b)

Abbildung 3.3. Filterung einer Höhenkarte: (a) Ungefilterte Höhenkarte. Im Bereich oben links sind deutliche Löcher in der Karte zu erkennen, die durch uninitialisierte Zellen zustande kommen. (b) Mit einem Median-Filter verbesserte Höhenkarte. Die Löcher konnten zufriedenstellend geschlossen werden, so dass die Oberfläche des Objekts deutlich zu erkennen ist.

Kapitel 4

Verhaltenskarten

Die Pfadplanung in dreidimensionalem Terrain ist insbesondere dann schwierig, wenn dieses komplexe Strukturen erhält, die der Roboter überwinden soll, da es dazu notwendig ist, dass der Roboter sich seines Kontexts explizit bewußt ist. Eine rein zweidimensionale Pfadplanung, die nicht beachtet, über welche Art von Umgebung der geplante Pfad führt, also nur zwischen befahrbar und Hindernis entscheidet, wird deshalb entweder Pfade produzieren, die die Möglichkeiten des Roboter nicht ausnutzen, also überwindbare Hindernisse nicht befahren, oder aber unbefahrbare Pfade produzieren. Dies liegt daran, dass eine zweidimensionale Pfadplanung in der Regel mehr Freiheitsgrade hat als eine Pfadplanung, die Hindernisse berücksichtigt. Die ist zum Beispiel der Fall, wenn der Pfad über ein Hindernis führen soll, das nur in einem bestimmten Winkelbereich überfahren werden kann. Weiterhin stellt sich das Problem, dass der Roboter auch geeignete Aktionen ausführen muss, um ein Hindernis zu überwinden. Rein verhaltensbasierte Ansätze sind auf mittelschwerem Terrain, wie zum Beispiel Wiesen oder Geröllhaufen durchaus erfolgreich, können aber sehr steile Hindernisse, wie Paletten oder Treppen nicht zuverlässig überwinden.

Eine mögliche Lösung des Problems wäre die explizite Handlungsplanung des Roboters in allen Freiheitsgraden der Bewegung, die zum Beispiel bei den Lurker-Robotern in den zwei Kettenantrieben links und rechts, sowie der Position der Arme vorne und hinten, bestehen. Dies scheint aus zwei Gründen unrealistisch. Erstens wird der Suchraum bei so einem Ansatz sehr schnell zu groß, um effektiv berechenbar zu sein. Verbesserungen durch Heuristiken sind hier durchaus realistisch, gerade bei Fahrt auf flachem Grund, allerdings ist insbesondere beim Überwinden von Hindernissen eine gezielte Suche nur schwer möglich. Der zweite Grund, der eine solche Planung in der Realität zum Scheitern verurteilt, besteht darin, dass gerade beim Überwinden von Hindernissen die Tiefe, bis zu der geplant werden kann, weil die Diskrepanz zwischen dem, was geplant wurde, und dem, was der Roboter wirklich ausführt, zu groß ist, sehr gering ist. Das kann dazu führen, dass ein Plan auf einem realen System gar nicht ausführbar ist, da eine Vorhersage des Zustands

des Roboters nach einer Aktion nur sehr unzuverlässig ist.

Da es das Ziel ist auf einer Karte Pfadplanung zu betreiben, die die speziellen Fähigkeiten des Roboters zum Überwinden von verschiedenartigen Hindernissen beachtet, soll die Planung auf *Verhaltenskarten* erfolgen. Im Vergleich zu konventionellen Karten, wie zum Beispiel *Occupancy Maps* [7] enthalten diese zusätzlich Kontextinformationen, die zum Überqueren von Hindernissen benötigt werden. Dadurch ist es möglich vorhandene Fähigkeiten eines Roboters wie zum Beispiel das Überfahren von Paletten oder Rampen in den Planungsprozess zu integrieren. Im Gesamtsystem werden diese damit sowohl in der Planung eingesetzt, wie auch in der Aktionsausführung.

Definition von Fähigkeitsbeschreibungen Damit verschiedenartige Fähigkeiten einfach in das Gesamtsystem integriert werden können, werden Verhaltenskarten automatisch aus einer Höhenkarte und einer Menge von Fähigkeitsbeschreibungen erstellt. Dazu enthalten diese eine Menge von *Fuzzy-Regeln*, die zur Klassifikation der entsprechenden Struktur benötigt werden, eine Menge *qualitativer räumlicher Beziehungen*, die Vorabbedingungen zur Verhaltensauswahl beschreiben, eine *Kostenfunktion* für die Pfadplanung und eine *Funktion zur Hindernisüberwindung*. Abbildung 4.1 stellt die Integration von Verhaltenskarten ins Gesamtsystem dar. Die Vorabklassifikation der Höhenkarte dient dazu, eine simple Unterscheidung in *Boden*, *Wand* und *Unentscheidbar* mit Hilfe der *Fuzzy-Regeln* vorzunehmen. Diese werden bei der Klassifikation der unentscheidbaren Regionen als Merkmale für *Markov Random Fields* wieder benutzt. Die *räumlichen Beziehungen* beschreiben unter welchen Bedinungen Hindernisse überwunden werden können und schränken so die Freiheitsgrade der Planung ein. Es ist dann möglich herkömmliche A*-Suche zur Pfadplanung zu nutzen. Die allgemeine Definition einer Fähigkeitsbeschreibung ermöglicht es, verschiedene Fähigkeiten des Roboters einfach in das Gesamtsystem zu integrieren, ohne dass dafür jeweils spezialisierte Algorithmen geschrieben werden müssen und bietet somit eine gute Basis zur Erweiterung des hier vorgestellten Systems.

Dieses Vorgehen wird in den nächsten Abschnitten nun näher erläutert.

4.1 Erstellen von Verhaltenskarten

Das Erstellen von Verhaltenskarten aus Höhenkarten mittels einer Menge von Fähigkeitsbeschreibungen geschieht in drei Schritten. Zuerst wird dazu eine Vorabklassifikation in Echtzeit vorgenommen und unentscheidbare Regionen werden identifiziert. Danach werden diese Regionen mittels eines Lernalgorithmus klassifiziert, um so bestimmte Hindernisklassen zu detektieren. Anschließend wird mit Hilfe der

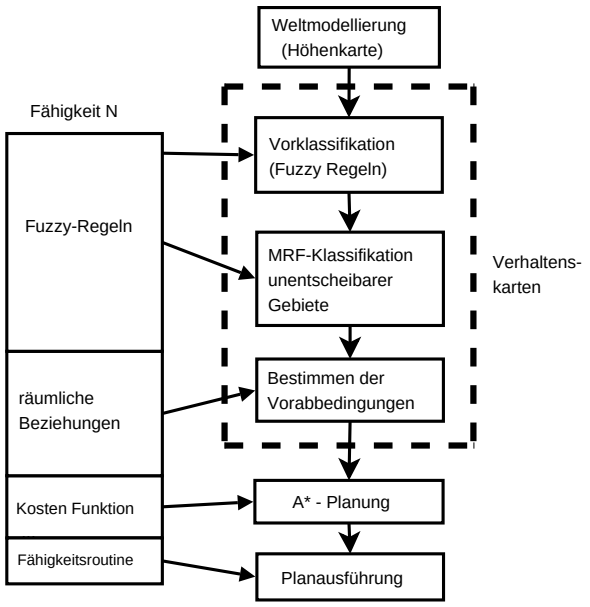


Abbildung 4.1. Konstruktion und Anwendung von Verhaltenskarten im Gesamtsystem.

räumlichen Beziehungen festgestellt, an welchen Stellen eine bestimmte Aktionsausführung möglich ist und welche Vorabbedingungen dazu notwenig sind. Diese Informationen werden in die Verhaltenskarte eingetragen und anschließend in der Planung und Aktionsausführung benutzt.

4.1.1 Fuzzy-Regeln zur Merkmalsauswahl

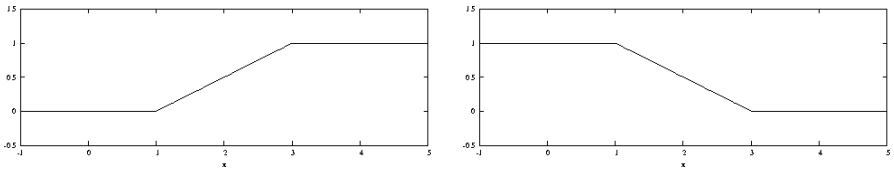
Ein erster Schritt im Klassifikationsprozess besteht in der Merkmalsauswahl. Diese dient dazu aus den absoluten Höheninformationen der Karte Merkmale zu extrahieren, die für eine Klassifikation mehr Aussagekraft haben. Um diese Merkmale zu repräsentieren, werden Parameter wie zum Beispiel der Höhenunterschied mit Hilfen von Funktionen, wie sie aus der Fuzzy Logic bekannt sind, auf den Bereich $[0, 1]$ skaliert, wobei 1 signalisiert, dass das Merkmal voll aktiv ist und 0 dafür steht, dass das Merkmal gar nicht vorhanden ist. Dadurch kann ein Merkmal nicht nur inaktiv oder aktiv sein, sondern Übergangsbereiche sind modellierbar. Grundsätzlich setzen sich alle hier verwendeten Merkmale aus den Funktionen $StraightUp$ (Gleichung 4.1) und $StraightDown$ (Gleichung 4.2) zusammen, die in Abbildung 4.2 dargestellt sind. $StraightUp(x, a, b)$ und $StraightDown(x, a, b)$ liefern dabei einen Wert zwischen 0 und 1, wobei x der Eingabewert ist und a den Anfang, sowie

b das Ende des Anstiegs, beziehungsweise des Abstiegs bezeichnet.

$$StraightUp(x,a,b) = \begin{cases} 0 & \text{falls } x < a \\ \frac{x-a}{b-a} & \text{falls } a \leq x \leq b \\ 1 & \text{falls } x > b \end{cases} \tag{4.1}$$

$$StraightDown(x,a,b) = 1 - StraightUp(x,a,b) \tag{4.2}$$

Mit Hilfe dieser Funktionen ist es nun möglich verschiedene Merkmale konkret anzugeben.



$StraightUp(x,1,3)$ $StraightDown(x,1,3)$

Abbildung 4.2. Graphen der Funktionen $StraightUp(x,1,3)$ und $StraightDown(x,1,3)$. Der Parameter x wird im Bereich zwischen 1 und 3 langsam aktiviert, bzw. deaktiviert.

Parameter, die für eine Zelle i berechnet werden, sind einerseits, basierend auf den jeweils aus dem Kalman-Filter der j-ten Zelle gesampleten Höhen h_j, der maximale Höhenunterschied δh_i in mm (Gleichung 4.3) und andererseits aus dem normalisierten Punktnormalenvektor $\mathbf{n_i}$ der Zelle i die „Schiefe“ der Zelle α_i in Grad (Gleichung 4.4), die als Winkelabweichung zum Normalenvektor $(0,1,0)^T$ der xz-Ebene definiert ist.

$$\delta h_i = \max_{j \text{ ist Nachbarzelle von } i} |h_i - h_j| \tag{4.3}$$

$$\alpha_i = (0,1,0)^T \cdot \mathbf{n_i} = n_{i_y} \tag{4.4}$$

Daraus lassen sich zum Beispiel folgende Merkmale erzeugen:

- Bodenhöhe $= StraightDown(\delta h_i, 15, 40)$
- Hindernishöhe $= StraightUp(\delta h_i, 25, 80) \cdot StraightDown(\delta h_i, 200, 350)$
- Wandhöhe $= StraightUp(\delta h_i, 200, 300)$
- Rampenwinkel $= StraightUp(\alpha_i, 3, 25) \cdot StraightDown(\alpha_i, 25, 40)$
- Steiler Winkel $= StraightUp(\alpha_i, 25, 90)$

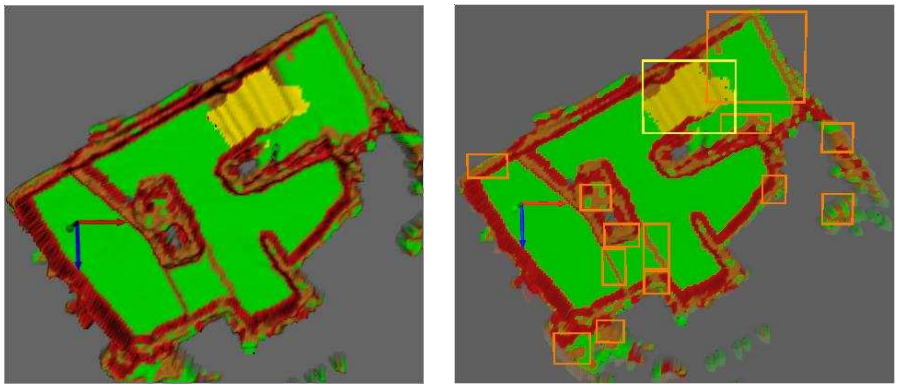
Mit Hilfe dieser Merkmale soll nun ein einfacher Klassifikator erzeugt werden. Dieser richtet sich nur danach, welches Merkmal aktiv ist und unterscheidet dementsprechend die Klasen *Boden*, *Wand* und *Unentscheidbar*, falls weder Boden, noch Wand aktiv sind. Der Vorteil dieser einfachen Klassifikation besteht darin, dass sie grundsätzlich akzeptable Ergebnisse liefert und dabei relativ schnell ist, das heißt konkret, dass diese Klassifikation in Echtzeit auf die Karte anwendbar ist. Es ist allerdings mit diesem Verfahren nicht möglich mit zufriedenstellender Genauigkeit Hindernisse wie zum Beispiel Rampen zu erkennen. Die grundsätzliche Unterscheidung zwischen Wand und Boden erlaubt es dem Roboter dennoch schon mit diesen Informationen zu planen und soll deshalb aufgrund der schnellen Laufzeit in jedem Zyklus als Vorklassifikation angewandt werden.

Im nächsten Abschnitt wird erläutert, wie aus den als *Unentscheidbar* erkannten Gebieten verschiedene Hindernisklassen identifiziert werden, sowie in klassifizierten Gebieten Übergänge erkannt werden. Da das dort angewandte Verfahren allerdings nicht in Echtzeit auf die gesamte Karte angewandet werden kann, ist es also noch nötig aus der Menge an klassifizierten Zellen die Gebiete festzustellen, in denen diese liegen. Dazu soll ein Algorithmus aus dem Computersehen [4] benutzt werden. Ursprünglich ist dieser dazu gedacht aus YUV-Bildern mittels Grenzwerten Farbbereiche in bestimmten YUV-Intervallen, die einer Farbklasse entsprechen, zu identifizieren. Aus der klassifierten Karte wird dazu ein Bild mit den Dimensionen der Karte erzeugt, wobei die Farbwerte der Pixel den jeweiligen Klassen zugeordnet werden. Der Algorithmus wird mit diesen Farbwertzuordnungen für die Klassen initialisiert und liefert als Ausgabe zu diesem Bild eine Liste von Bereichen, die segmentiert wurden, sowie deren Größe. Abbildung 4.3 zeigt das Resultat der Gebietserkennung auf einer Karte.

4.1.2 Klassifikation durch Lernen von Markov Random Fields

Die Vorklassifikation aus Abschnitt 4.1.1 beruht nur auf lokalen Daten bezüglich einer Zelle. Reale Umgebungen und Hindernisse sind allerdings fast immer deutlich größer und umspannen eine Vielzahl von Zellen, so dass sich eine Klassifikation erst aus der Kombination dieser Zellen ergeben kann. Aus diesem Grund wurde eine Klassifikation mit Markov Random Fields (siehe Abschnitt 2.4) vorgenommen. Diese basieren auf einem Graphen und können Verknüpfungseigenschaften zwischen den Knoten des Graphens wiedergeben und somit eine Klassifikation vornehmen, die nicht nur die Zellen einzeln betrachtet, sondern bei der Klassifikation jeder Zelle deren Umgebung miteinbezieht.

Im Vergleich zu handgeschriebenen Algorithmen zur Klassifikation bietet dieses Vorgehen mehrere Vorteile. Zum einen ist es wesentlich einfacher eine neue Hindernisklasse zu implementieren, da zum Ermitteln der Gewichte ein Lernalgorithmus



(a) (b)

Abbildung 4.3. Erkennung interessanter Regionen: (a) Klassifizierte Höhenkarte. Bodenregionen sind grün gekennzeichnet und Wände erscheinen rot. Möglicherweise interessante Bereiche sind mit braun (Hindernis) und gelb (Rampe) markiert. (b) Die Rechtecke markieren die erkannten zusammenhängenden Gebiete.

verwendet wird. Dadurch müssen in der Fähigkeitsbeschreibung lediglich *Fuzzy-Regeln*, die für dieses Hindernis aussagekräftig sind, beschrieben und klassifizierte Daten bereitgestellt werden. Zum anderen ist ein handgeschriebener Klassifikator in der Regel sehr speziell, nicht nur für die Hindernisklasse, sondern eventuell sogar für ein bestimmtes Hindernis. Bei der Verwendung eines Lernalgorithmus lässt sich eine übermäßige Spezialisierung(„Overfitting“) vermeiden, indem man die Menge an unterschiedlichen Lerndaten vergrößert.

Die für das MRF benötigten Merkmalsvektoren x_i der Knoten setzen sich aus den *Fuzzy-Regeln* der Fähigkeitsbeschreibung, die schon in Abschnitt 4.1.1 für die Vorklassifikation definiert wurden, zusammen. Zur Verknüpfung der Informationen zwischen Zellen benötigt dass MRF noch Merkmalsvektoren x_{ij}, die einer Kante zwischen einer Zelle i und einer Zelle j zugeordnet werden. Dabei verwendete Parameter sind die Zellhöhe h_i, beziehungsweise h_j und die normalisierten Punktnormalenvektoren $\mathbf{n_i}$ und $\mathbf{n_j}$, aus denen der Höhenunterschied δh (Gleichung 4.5) und der Richtungsunterschied $\delta\alpha$ (Gleichung 4.6) berechnet werden.

$$\delta h = |h_i - h_j| \tag{4.5}$$

$$\delta\alpha = \cos^{-1}(\mathbf{n_i} \cdot \mathbf{n_j}) \tag{4.6}$$

Daraus lassen sich für Kanten zum Beispiel folgende Merkmale berechnen:

- BodenUnterschied $= StraightDown(\delta h, 15, 40)$
- HindernisDelta $= StraightUp(\delta h, 25, 80) \cdot StraightDown(\delta h, 200, 350)$
- WandUnterschied $= StraightUp(\delta h, 200, 300)$

- ÄhnlicherWinkel $= StraightDown(\delta\alpha, 0, 15)$

Das *ÄhnlicherWinkel*-Merkmal soll dazu dienen Zellen mit Normalenwinkeln, die im Winkelbereich einer Rampe liegen und in die gleiche Richtung zeigen, von Zellen, die auch im Winkelbereich einer Rampe liegen, aber in eine andere Richtung zeigen, wie es zum Beispiel bei kleinen Erhebungen auf nicht ganz flachem Boden häufig vorkommt, zu unterscheiden.

Zur Klassifikation der als *Unentscheidbar* vorklassifizierten Regionen wird für jede Region ein Graph für das MRF aus der Karte erzeugt. Dabei wird die Struktur der Karte direkt umgesetzt, das heißt: Jede Zelle der Karte ist ein Knoten im MRF-Graph und die Kanten des Graphen bestehen jeweils zwischen den Nachbarzellen auf der Karte. Der Hauptgrund dafür, dass die MRF-Klassifikation nur auf Regionen der Karte und nicht auf der gesamten Karte vorgenommen wird, liegt darin, dass die Laufzeiten einer solchen Klassifikation je nach Größe des Graphen im Bereich von mehreren Sekunden liegen. Deshalb wird die MRF-Inferenz im Hintergrund berechnet. Es soll allerdings verhindert werden sehr ähnliche Regionen oftmals zu klassifizieren, da dadurch unnötig viel Rechenzeit verbraucht wird.

Sei dazu ein rechteckiges Gebiet durch die zwei Ecken (x_1, y_1), (x_2, y_2) definiert, wobei jeweils gelten soll, dass $x_1 \leq x_2$ und $y_1 \leq y_2$. Für alle Gebiete $(x_1^i, y_1^i), (x_2^i, y_2^i)$, die schon zur Klassifikation anstehen, beziehungsweise die schon durch das MRF klassifiziert wurden, wird für ein zweites Gebiet $(x_1^n, y_1^n), (x_2^n, y_2^n)$, das klassifiziert werden soll, das Schnittgebiet $(x_1^s, y_1^s), (x_2^s, y_2^s)$ gegeben durch Gleichung 4.7 und Gleichung 4.8.

$$(x_1^s, y_1^s) = \big(\max(x_1^i, x_1^n), \max(y_1^i, y_1^n)\big) \tag{4.7}$$

$$(x_2^s, y_2^s) = \big(\min(x_2^i, x_2^n), \min(y_2^i, y_2^n)\big) \tag{4.8}$$

Der Flächenanteil f_s des Schnittgebiets an dem neuen Gebiet $(x_1^n, y_1^n), (x_2^n, y_2^n)$ aus Gleichung 4.9 gibt ein Maß dafür an, wieviel von dem Gebiet schon klassifiziert ist. Wenn die Gebiete sich nicht schneiden, gilt dabei, dass $f_s := 0$.

$$f_s = \frac{(x_2^s - x_1^s) \cdot (y_2^s - y_1^s)}{(x_2^n - x_1^n) \cdot (y_2^n - y_1^n)} \tag{4.9}$$

Liegt dieser Anteil für alle $(x_1^i, y_1^i), (x_2^i, y_2^i)$ unter einer bestimmten Grenze c, also gilt, dass $f_s < c$, dann ist ein genügend großer Anteil des Gebietes noch nicht klassifiziert und soll somit klassifiziert werden. Dies umfasst selbst bei $c = 1$ den Spezialfall $f_s = 1$, das heißt, dass das gesamte Gebiet schon klassifiziert wurde. Aktuell wird $c = 0.8$ verwendet.

4.1.3 Kodierung von Vorabbedingungen

Im Gegensatz zu flachem Boden ist ein klassifiziertes Hindernis, wie zum Beispiel eine Rampe, nicht unbedingt von jeder Seite und jeder Richtung befahrbar. Deshalb müssen Übergänge, die auf das Hindernis führen, detektiert werden. Des weiteren ist es notwendig Vorabbedingungen, die zu einer erfolgreichen Ausführung der Fähigkeitsroutine nötig sind, festzustellen. Dazu muss in jeder Fähigkeitsbeschreibung definiert werden, welche Art von Übergang diese bewältigen kann. Die geschieht mit qualitativen räumlichen Beziehungen, die hier formal durch eine Untermenge der Relationen aus *Allens Interval Kalkül* [1] gegeben sind. Um zum Beispiel einen Übergang von Boden zu einer Rampe zu beschrieben dient folgende Menge von Regeln: $\{B$ meets R, B starts G, R ends $G\}$. Abbildung 4.4 zeigt einen solchen Boden-Rampe Übergang, in dem keine anderen Intervalle vorkommen ausser *Boden* (B) und *Rampe* (R). Dabei steht G für den Interval, der alle betrachteten Intervalle enthält.

Die Berechnung dieser Vorabbedingungen wird auf den in Abschnitt 4.1.1 berechneten und im vorherigen Abschnitt klassifizierten Gebieten vorgenommen. Diese enthalten allerdings bisher nur eine Menge von klassifizierten Zellen. Es werden deshalb folgende Schritte vorgenommen: Zuerst werden Zellen identifiziert, die potentiell einen Übergang darstellen können. Danach werden Geraden aus diesen Zellen extrahiert und Start- und Endpunkte mittels der räumlichen Beziehungen identifiziert. Die so ermittelten Linien sollen im folgenden als *Übergangskanten* bezeichnet werden. Als letztes werden die Vorabbedingungen, das heißt Startpunkt und -richtung für die Aktionsausführung, berechnet.

Identifikation der möglichen Übergangszellen

Zur Identifikation der möglichen Übergangszellen werden aus der Menge aller Zellen in dem zu untersuchenden Gebiet die Zellen ausgewählt, die eine der *meets*-Regeln der Fähigkeitsbeschreibung erfüllen. Für das Beispiel der Rampe gibt es die Regel $\{B$ meets $R\}$. Damit würden alle *Boden*-Zellen, die eine *Rampe*-Zelle als Nachbarzelle haben und alle *Rampen*-Zellen, die eine *Boden*-Zelle als Nachbarzelle haben, ausgewählt. Dies bewirkt, dass alle Zellen ausgewählt werden, die eine Grenze zu einem anderen Gebiet darstellen, analog zur Definition von *Frontier-Zellen* [35], die eine Grenze zwischen unexplorierten und befahrbaren Gebieten definieren.

Bestimmen der Übergangskanten

Es wird die Annahme getroffen, dass eine *Übergangskante* in der Regel durch eine Gerade dargestellt werden kann. Deshalb werden nun aus der Menge von aus-

gewählten Zellen mittels der Hough-Transformation (siehe Abschnitt 2.3) Geraden extrahiert. Falls es für eine neue Art von Hindernissen erforderlich ist andere geometrische Formen zu erkennen, lässt sich auch die allgemeine Hough-Transformation verwenden, die zum Beispiel Kreise erkennen kann.

Die Hough-Transformation liefert allerdings nur Geraden im Raum. Es ist deshalb nötig aus dieser Geraden eine Linie mit Anfangs- und Endpunkt zu bestimmen, die die *Übergangskante* darstellt. Dazu wird für jede Zelle auf der Geraden überprüft, ob diese die qualitativen räumlichen Beziehungen aus der Fähigkeitsbeschreibung erfüllt und damit ein Punkt eines Übergangs sein kann. Eine Linie mit der Länge entsprechend der Roboterlänge wird dazu in Richtung des möglichen Übergangs, also senkrecht zur untersuchten Geraden, durch die zu untersuchende Zelle gelegt. Entlang dieser Linie werden die Intervalle abhängig von den Klassen bestimmt, wobei ein Intervall jeweils an der Zelle der entsprechenden Klasse beginnt und erst dann endet, wenn sich entlang der Linie eine Zelle einer anderen Klasse befindet. Die Überprüfung der räumlichen Beziehungen auf diesen Intervallen lässt sich durch die Beschränkung auf die *meets*, *starts* und *ends* Beziehungen einfach berechnen, so dass es hierfür nicht notwendig ist allgemeingültige Algorithmen, wie zum Beispiel Backtracking mit Path-Consistency zu benutzen. Wenn entlang der Geraden aus der Hough-Transformation Zellen über mindestens eine Roboterbreite die Bedingungen erfüllen, so stellen diese eine *Übergangskante* dar.

Berechnen von Vorabbedingungen

Zu den *Übergangskanten* sollen nun Startpunkt (x^*, y^*) und Startrichtung θ^* als Vorabbedingungen für die Aktionsausführung berechnet werden. Dazu wird die Startrichtung θ^* in Richtung des Übergangs gelegt, also senkrecht zur Übergangskante, und die Startposition wird vom Mittelpunkt der Übergangskante aus mit der Entfernung von einer Roboterlänge vor die Übergangskante projiziert. Des weiteren wird als Parameter der vorzeichenbehaftete mittlere Höhenunterschied zwischen den beiden Seiten der Übergangskante berechnet. Dieser dient Fähigkeitsroutinen dazu, um zum Beispiel zu bestimmen, ob der Übergang aufwärts oder abwärts gerichtet ist, und beeinflusst dadurch deren Ausführung. Jeder Kartenzelle, die sich entlang dieser Übergangskante befindet, werden alle berechneten Parameter der *Übergangskante*, sowie die entsprechende Fähigkeitsroutine mitgeteilt. Dadurch entsteht die *Verhaltenskarte*.

Das Verfahren wird in Abbildung 4.4 und Abbildung 4.5 am Beispiel einer Rampe und einer Palette visualisiert. Dabei ist in (a) jeweils eine Höhenkarte einer Rampe, beziehungsweise einer Palette zu sehen. (d) zeigt die klassifizierte Höhenkarte mit den extrahierten Gebieten als Rechtecke. Dabei sind Wände rot, Bodenzellen grün, Rampen gelb und Hinderniszellen, wie zum Beispiel Paletten braun. Abbil-

dung 4.4(b) zeigt die Zellen, die aus dem Gebiet der Rampe als Eingabe für die Hough-Transformation dienen und Abbildung 4.5(b) die Zellen, die aus dem rechten Gebiet der Palette für die Hough-Transformation ausgewählt wurden. In (e) ist dann in cyan jeweils die von der Hough-Transformation zurückgegebene Gerade in dem jeweiligen Gebiet zu sehen. Die Übergangskanten sind in (f) rot dargestellt. Weiterhin kann man den Startpunkt zur Aktionsausführung (blau), sowie die Startrichtung (gelb), erkennen. In (c) ist dann jeweils die korrespondierende Verhaltenskarte zu sehen, wobei Zellen die ein Rampenverhalten erfordern gelb sind und Palettenverhalten mit braunen Zellen markiert sind. Nicht eingefärbte Zellen implizieren Bodenfahrverhalten.

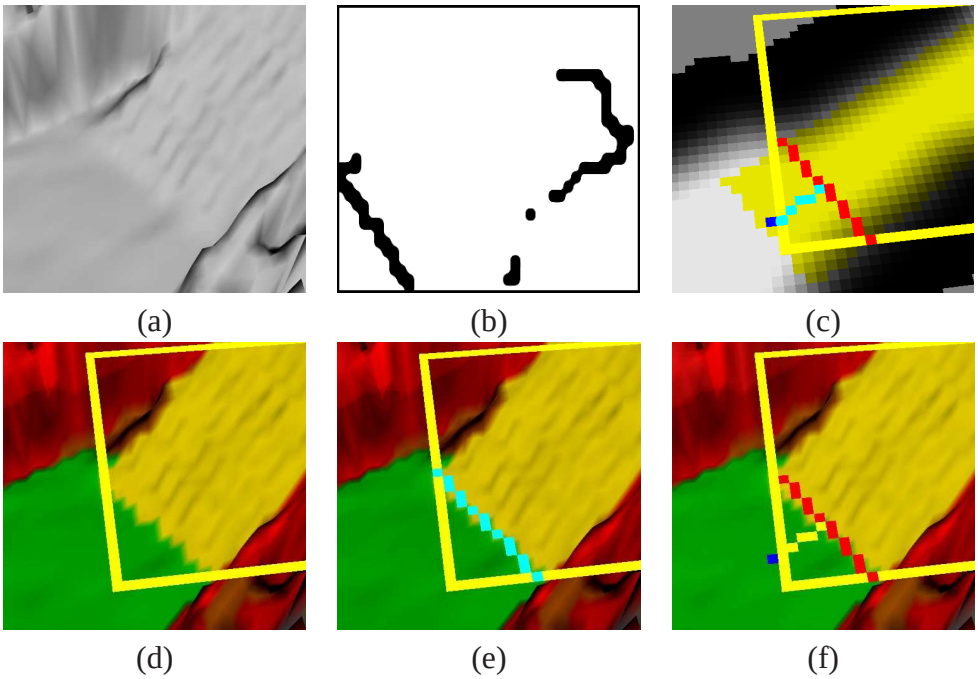


Abbildung 4.4. Erkennung der *Übergangskante* zu einer Rampe und Berechnung der Vorabbedingungen zur Rampenfahrt.

4.2 Zielauswahl

Die Aufgabe des Roboters soll in der Exploration von unbekanntem Gebiet bestehen. Um in der dem Roboter aktuell bekannten Karte navigieren zu können, scheint es deshalb nur sinnvoll den Roboter in unerforschte Regionen zu steuern. Die Zielauswahl beruht deshalb auf dem bekannten Konzept von *Frontier-Zellen* [35]. Eine Frontier-Zelle ist dabei definiert als eine Zelle der Karte, die traversierbar ist und

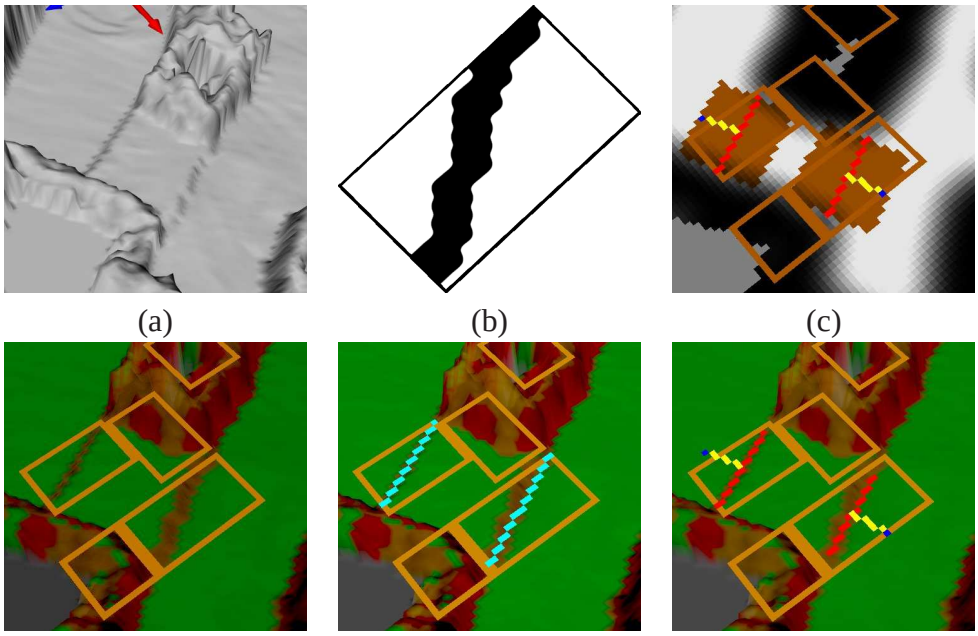


Abbildung 4.5. Erkennung der *Übergangskanten* einer Pallette und Berechnung der Vorabbedingungen zum Befahren einer Palette.

unter deren Nachbarn sich mindestens eine uninitialisierte Zelle befindet. Natürlicherweise befinden sich solche Zellen an der Grenze der Karte zu unbekanntem Terrain, so dass der Roboter sich in dieses bewegt, wenn er *Frontier-Zellen* anfährt.

Zuerst werden dazu alle Zellen, die die Definition einer Frontier-Zelle erfüllen, gefiltert, so dass nur noch eine Teilmenge dieser Frontier-Zellen übrig bleibt, bei der gilt, dass je zwei Zellen in dieser Menge einen Mindestabstand von 20 cm nicht unterschreiten, damit die nachfolgende Auswahl nicht auf der gesamten Anzahl aller Frontier-Zellen arbeiten muss. Zellen, die sehr nahe aneinanderliegen, entscheiden sich nicht grundsätzlich, so dass der Informationsverlust durch die Filterung nicht sehr groß ist. Danach wird jeder Zelle ein Nutzen u zugewiesen, der sich nach Gleichung 4.10 berechnet, wobei $\delta\theta$ den Betrag des Winkels bezeichnet unter dem der Roboter die Zelle sieht, so dass eine Zelle mit $\delta\theta = 0$ zum Beispiel direkt vor dem Roboter liegt.

$$u = (StraightDown(\delta\theta, 0, 180))^2 \tag{4.10}$$

Die Funktion $StraightDown$ ist dabei eine Fuzzy-Regel, die in Gleichung 4.2 definiert ist, und hier im wesentlichen eine lineare Interpolation darstellt. Ziel dieser Nutzenberechnung ist es, dass der Roboter bevorzugt zu Zellen fährt, die vor ihm liegen, so dass dieser möglichst wenig drehen muss und Oszillationseffekte vermieden

werden.

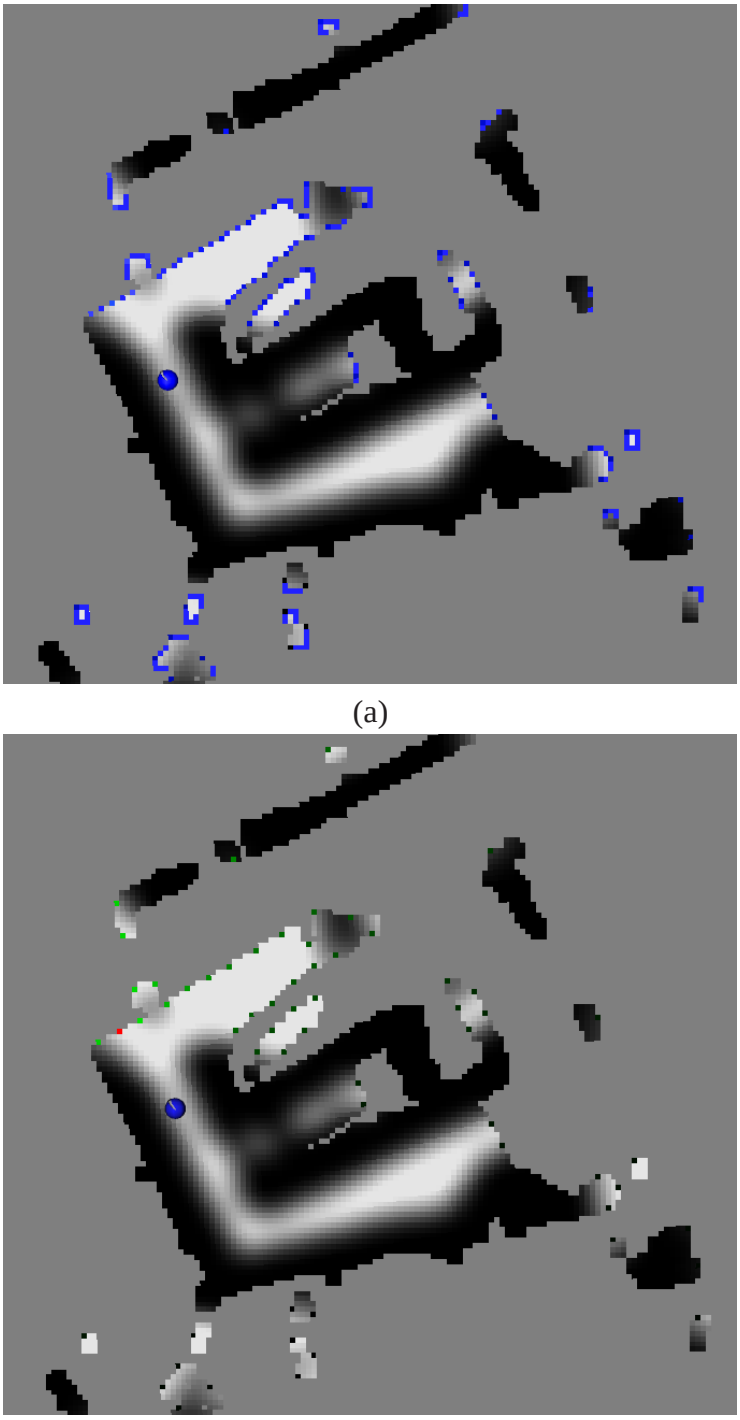
Grundsätzlich soll nun eine erreichbare Frontier-Zelle angefahren werden, die maximalen Nutzen hat. Dadurch ergibt sich ein robustes Explorationsverfahren, das sicherstellt, dass das gesamte erreichbare Gebiet exploriert wird. Das Verfahren ist als Greedy-Algorithmus offensichtlich nicht unbedingt optimal hinsichtlich der Explorationszeit. Bei der Exploration auf schwierigem Terrain gibt es allerdings andere Optimalitätskriterien als zum Beispiel bei der Exploration mit mehreren Robotern auf flachem Boden. So ist das Überqueren eines Hindernisses zwar Bestandteil der Exploration, sollte allerdings möglichst nicht ausgewählt werden, solange es geeignete Ziele gibt, die ohne Hindernisse zu überqueren erreichbar sind, da die Fehlerwahrscheinlichkeit auf einem Hindernis deutlich größer als auf flachem Grund ist. Diese Eigenschaft wird durch die Kostenfunktion, die zu den Fähigkeitsbeschreibungen gehört, wiedergegeben. Die Zielauswahl akzeptiert dazu zuerst nur Pläne, die unter einer bestimmten Kostenschwelle liegen, um so die Exploration über Boden der Exploration über Hindernisse vorzuziehen, selbst, wenn diese direkt vor dem Roboter liegen. Erst wenn keine Ziele unter dieser Kostenschwelle mehr vorhanden sind, werden auch Ziele mit höheren Kosten, also hinter Hindernissen, akzeptiert. Dies bewirkt, dass der Roboter ein durch Hindernisse abgegrenztes Gebiet erst vollständig exploriert, bevor er diese überquert.

4.3 A*-Planung

Die Planung zu einem Ziel beruht auf dem bekannten A*-Algorithmus (siehe Abschnitt 2.4) für Graphensuche in zellenbasierten Karten. Dazu ist es nun noch nötig Start- und Zielbedingungen, Kosten- und Nachfolgerfunktion, sowie eine Heuristik zu definieren. Die Startzelle ist dabei einfach die Zelle, auf der der Roboter sich gerade befindet und die Zielbedingung ist dann erfüllt, wenn die zu testende Zelle die gleichen Koordinaten wie die Zielzelle hat.

Als Nachfolger einer traversierbaren Zelle kommen alle traversierbaren Zellen in Frage, die sich direkt neben der Zelle befinden. Dabei ist die Traversierbarkeit durch die Verhaltenskarte und damit die Fähigkeiten des Roboters bestimmt. Das geschieht dadurch, dass es zu einer Zelle nur Nachfolger einer anderen Klasse gibt, wenn sich an dieser Position eine entsprechende *Übergangskante* befindet.

Die Kostenfunktion der A*-Suche ist auf Bodenzellen grundsätzlich über Werte zwischen 0 und 1 gegeben, die in Abschnitt 4.3.1 näher beschrieben werden. Um das langsamere und auch schwierigere Vorankommen auf Hindernissen wiederzugeben, haben Fähigkeitsbeschreibungen die Möglichkeit eigene Kostenfunktionen zu definieren, die deutlich größere Kosten liefern können als die Kosten für normale Fahrt. Die für die A*-Suche verwendete Heuristik ist die euklidische Distanz



(a)

(b)

Abbildung 4.6. Auswahl von Frontier-Zellen: (a) Alle möglichen Frontier-Zellen (blau). (b) Die ausgewählten Frontier-Zellen mit einem Mindestabstand von 20 cm (grün). Je höher der Nutzen einer Frontier-Zelle ist, desto heller ist diese eingefärbt. Die Frontierzelle mit dem größten Nutzen ist rot markiert. Der Roboter schaut dabei in einem Winkel von ca. 25° nach links oben.

$h = \sqrt{\delta x^2 + \delta y^2}$. Durch die stark erhöhten Kosten bei Hindernissen, unterschätzt diese Heuristik eventuell sehr stark, was sich ungünstig auf die Laufzeiten auswirkt. Um aber die Optimalität der Planung zu gewährleisten, wurde die Heuristik nicht angepasst. Bei der praktischen Anwendung hat sich gezeigt, dass die Laufzeiten kein Problem für die Echtzeitfähigkeit des Gesamtsystems darstellen.

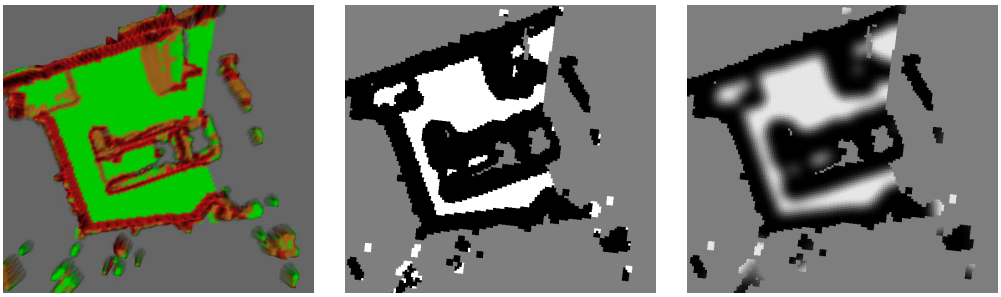
4.3.1 Kartenvorverarbeitung

Da es sich bei dem Roboter um ein reales System handelt, ist die exakte Ausführung eines Plans nicht immer gewährleistet. So kann es durch Ungenauigkeiten in der Lokalisierung oder in der Ansteuerung dazu kommen, dass der Roboter nicht genau dem geplanten Pfad folgt. Da dies insbesondere in der Nähe von Wänden und Engstellen kritisch ist, wird deshalb eine Vorverarbeitung der Karte vorgenommen. Dabei werden zuerst die nicht traversierbaren Zellen mit Kosten von 1 initialisiert und die befahrbaren Zellen mit Kosten von 0. Um die Ausmaße des Roboters zu berücksichtigen, werden nun alle Zellen, die sich näher als eine halbe Roboterbreite an einer nicht traversierbaren Zelle befinden, ebenfalls auf Kosten von 1 gesetzt. Dies ermöglicht es nur auf einzelnen Zellen zu planen, so dass man während der Planung auf Kollisionsabfragen verzichten kann. Anschließend wird die Karte mit einem Gauss-Kern weichgezeichnet. Dazu legt man eine Gaussglocke über jede Zelle und gewichtet die Kosten aller Zellen im Umkreis der Glocke mit dem Wert der Gauss-Funktion, um so die neuen Kosten der Zelle zu erhalten. Dies hat den Effekt, dass Zellen in der Nähe von nicht traversierbaren Zellen deutlich höhere Kosten haben als weiter von Hindernissen entfernte Zellen. Dadurch wird erreicht, dass der kostengünstigste Pfad sich nicht direkt an Wänden befindet, wie es bei uniformen Kosten zu erwarten wäre.

In einem letzten Schritt werden die minimalen Kosten jeder Zelle auf $\frac{1}{10}$ gesetzt, um sicherzustellen, dass ein Pfad einer bestimmten Länge immer gewisse Minimalkosten besitzt und insbesondere Zellen mit Kosten 0 verhindert werden. Um zu erreichen, dass die verwandte Luftlinienheuristik zulässig ist, werden die Kosten mit dem Faktor 10 multipliziert, damit die Kosten zwischen zwei Zellen mindestens deren Abstand entsprechen. Für die A*-Planung stellt dies keinen Unterschied dar, da der konstante Faktor allen Zellen gleichermassen zukommt.

4.3.2 Speicherung explorierter Knoten

Beim hier vorliegenden Szenario ist es durchaus realistisch, dass die Pfadplanung in einem Zyklus mehrfach erfolgt, weil ein Ziel unerreichbar ist oder die geforderte Kostenschwelle nicht einhält (siehe 4.2). Mehrmaliges Aufrufen wäre aus Performanzgründen nicht erstrebenswert, wenn man jedesmal einen neuen A* Plan ge-

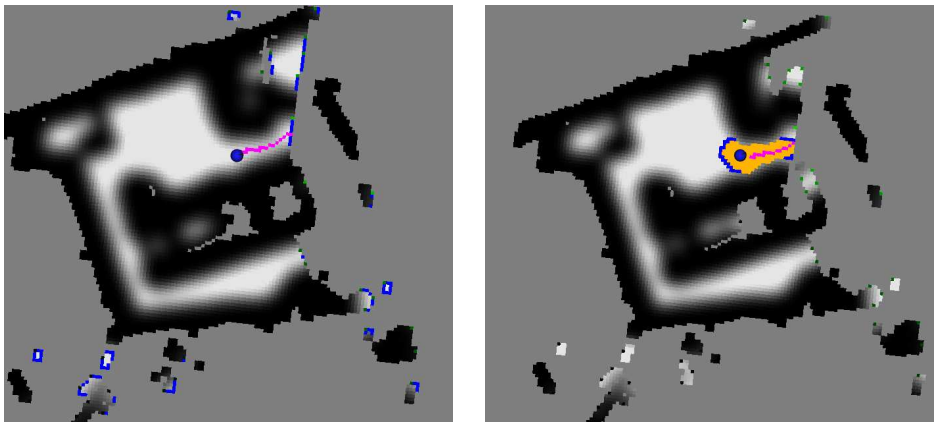


(a) (b) (c)

Abbildung 4.7. Die Schritte der Kartenvorverarbeitung: (a) Die klassifizierte Höhenkarte (grün - Boden, rot - Wand, braun - Hindernis). (b) Die entsprechende Karte mit gewachsenen Hindernissen (weiß - befahrbar, schwarz - nicht befahrbar). (c) Die mit einem Gauss-Kern weichgezeichnete Karte.

nerieren müsste, da gerade, wenn kein geeigneter Plan gefunden wurde, die Wahrscheinlichkeit, dass darauffolgende Aufrufe ebenfalls fehlschlagen, erhöht ist, weil diese meist ähnliche Ziele suchen.

Um dieses Problem zu umgehen, kann man sich hier eine Eigenschaft des A* zunutze machen. Diese besteht darin, dass der A*, sobald er den Zielknoten expandiert, gleichzeitig den optimalen Pfad zu diesem gefunden hat. Diese Eigenschaft gilt allerdings nicht nur für den Zielknoten, sondern für jeden Knoten, der expandiert wird. Beim graphenbasierten A* werden die expandierten Knoten in der sogenannten „Closed List“ gespeichert, um die Suche über Zykel im Graphen zu vermeiden. Wenn nun kein geeigneter Plan gefunden wurde, heißt das, dass entweder der Suchraum komplett expandiert ist oder die maximale Anzahl an Expansionen erreicht wurde. Dies bedeutet aber, dass bereits eine relativ große Closed List vorhanden ist, die schon optimale Wege zu vielen Knoten enthält. Verzichtet man nun nach jedem fehlgeschlagenen Planungsschritt während eines Zyklus auf das Löschen der Closed List, so kann man in einem nachfolgenden Aufruf die Planung mit dieser wieder initialisieren. Im einfachsten Fall besteht dann die Planung nur in einem Aufruf, der feststellt, ob der Zielknoten in der Closed List enthalten ist oder nicht. Dadurch ist es möglich nachfolgende Aufrufe in konstanter Zeit zu tätigen, so dass das mehrmalige Aufrufen des Planers bis ein geeigneter Plan gefunden wurde die Rechenzeit nicht wesentlich beeinflusst und diese insbesondere nicht linear mit der Anzahl der Aufrufe wächst.



(a) (b)

Abbildung 4.8. A* - Planung: (a) Ein mit A* generierter Plan (violett) zu der ausgewählten Frontier-Zelle. (b) Die expandierten Knoten des A*-Algorithmus (orange) und die Knoten, die sich noch in der Warteschlange des A* befinden (blau). Gut zu erkennen ist die durch die Heuristik gesteuerte zum Ziel gerichtete Suche.

4.4 Planausführung

Durch den Plan weiß der Roboter nun, wohin er fahren soll, aber noch nicht wie, das heißt welches Verhalten er ausführen soll. Die Verhaltensauswahl wird allerdings durch die Verwendung von Verhaltenskarten stark vereinfacht, da der Roboter das Verhalten wählt, das in der Verhaltenskarte für den aktuellen Ort eingetragen ist. Dies ist in der Regel ein Bodenfahrverhalten oder eine Hindernisroutine. Falls das Verhalten wechselt, also zum Beispiel eine Rampenroutine aufgerufen wird, wobei im letzten Zyklus noch Bodenfahrt ausgeführt wurde, ist es eventuell noch nötig Vorabbedingungen sicherzustellen, die für die korrekte Ausführung des neuen Verhaltens erforderlich sind. Dies ist auch relativ einfach möglich, da ja die notwendigen Vorabbedingungen auch in der Verhaltenskarte eingetragen sind. Der Roboter führt nun also das seinem Kontext entsprechend passende Verhalten aus. Im Moment besitzt der Roboter folgende autonomen Fähigkeiten: Bodenfahrt, Überqueren von Paletten, Rampenfahrt und Herauffahren von Treppen.

Das Bodenfahrverhalten ist aufgrund des relaisgesteuerten Kettenantriebs relativ simpel. Der Roboter kann lediglich drehen oder vorwärts, beziehungsweise rückwärts fahren, also insbesondere keine Kurven fahren. Um einem bestimmten Pfad zu folgen prüft der Roboter, ob seine aktuelle Richtung auf den Pfad ausgerichtet ist, und folgt diesem dann. Stimmt die aktuelle Roboterrichtung nicht mit dem Pfad überein, so dreht sich der Roboter dementsprechend. Um Oszillationen zu vermeiden, weil ein bestimmter Winkel nicht angefahren werden kann, ist es möglich die Motorlaufzeiten mit anzugeben. Diese werden proportional zum Winkelfehler mit

dem Pfad gewählt, was den Effekt einer Pulsweitenmodulation hat, und somit eine verringerte Rotationsgeschwindigkeit bewirkt.

4.4.1 Hindernisroutinen

Jeder Hindernisroutine ist das Überprüfen der Vorabbedingungen vorgeschaltet. Da die Pfadplanung keine Winkel beachtet, ist es durchaus realistisch, dass der aktuelle Roboterwinkel noch deutlich vom Startwinkel abweicht. Um diesen zu korrigieren, wird das gleiche Verhalten angewandt, dass auch bei Bodenfahrt den Roboter dreht. Da dies als Bestandteil der Hindernisroutine aufgefasst wird, ist es hier auch möglich Genauigkeiten anzugeben. So reicht den Verhalten, die sich auf eine Kante physikalisch ausrichten könnten (Palette und Treppe) hier ein relativ grober Winkel, während das Rampenfahrverhalten den Winkel sehr genau anfährt.

Die Ausführung der verschiedenen Hindernisroutinen beruht dann grundsätzlich auf einem gemeinsamen System. Als Sensoren zur Weltmodellierung benutzt dieses Nick- und Rollwinkel des Lagesensors, Winkelsensoren an den Armen und Kontaktsensoren an verschiedenen Stellen in den Armen. Aufgrund von Merkmalen, die aus diesen Sensordaten berechnet werden, läuft dann ein endlicher Automat, der verschiedene Aktionen parallel ausführen kann (zum Beispiel Vorwärtsfahren bei gleichzeitigem Absenken der hinteren Arme). Eine genauere Beschreibung des Systems und der Hardwaremodifikationen findet sich in [25]. Es soll nun auf den nächsten Seiten der grundsätzliche Ablauf dieser Aktionen gezeigt werden.

Herauf- und Herunterfahren einer Palette. Der Roboter startet unter der Annahme, dass sich vor ihm ein Hindernis in der Form einer Palette befindet, das es heraufzufahren gilt (Abbildung 4.9(a)). Zuerst fährt der Roboter an das Hindernis heran und prüft, ob beide Arme Kontakt mit dem Hindernis haben. Ist dies nicht der Fall, so fährt nur die Seite mit dem Arm vorwärts, der keinen Kontakt hat, was den Roboter gerade auf das Hindernis ausrichtet (Abbildung 4.9(b)). Abbildung 4.9(c) zeigt wie der Roboter sich mit dem Vorderarmen auf das Hindernis hebt und schließlich die Position in Abbildung 4.9(d) erreicht. Durch Vorwärtsfahren gelangt der Roboter auf das Hindernis. Zum Herabfahren hebt der Roboter sich leicht mit beiden Armen hoch (Abbildung 4.9(e)), um so beim Vorwärtsfahren die Kante zu bemerken. Dann werden die vorderen Arme nach unten gesenkt (Abbildung 4.9(f)) und die hinteren Arme nach oben gerichtet, um ein gleichmäßiges Herabfahren sicherzustellen. Am Ende werden die Arme wieder in die Fahrposition gebracht (Abbildung 4.9(g)), um die Weiterfahrt zu ermöglichen.

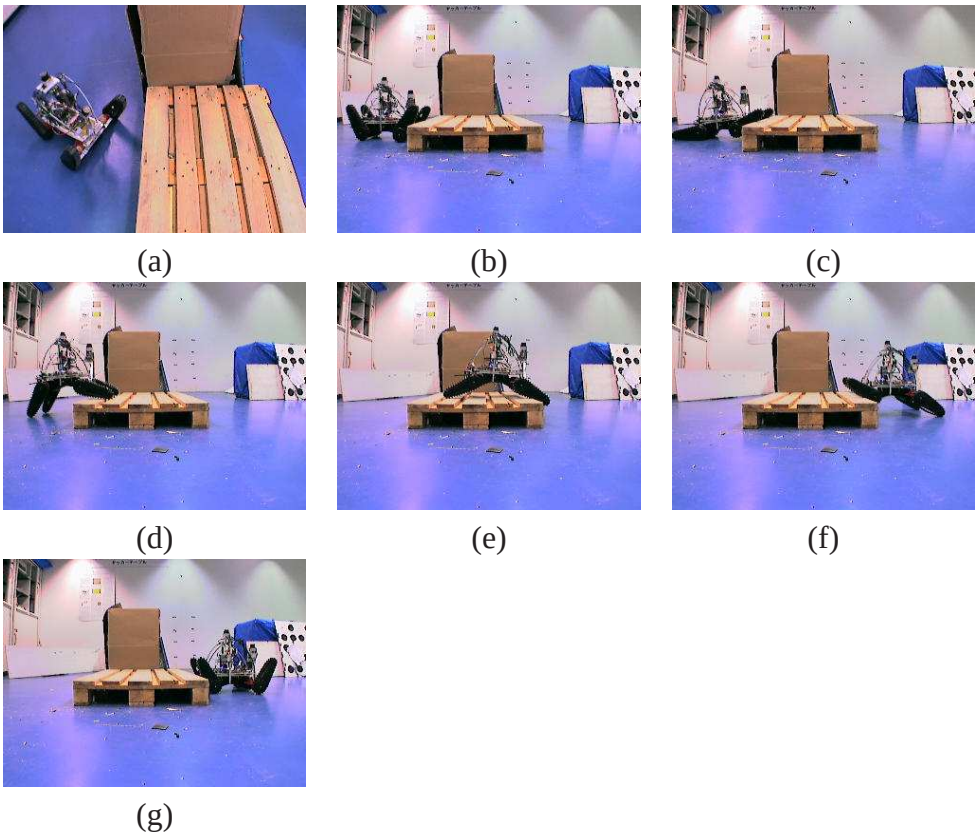


Abbildung 4.9. Überqueren einer Palette.

Rampenfahrt Vor der Rampenfahrt richtet sich der Roboter entsprechend der Vorabbedingungen auf die Rampe aus (Abbildung 4.10 (a) und (b)). Das Verhalten beginnt, indem der Roboter auf die Rampe fährt, bis er vollständig auf der Rampe ist (Abbildung 4.10 (c) und (d)). Dies wird durch einen geringen Gradienten des Nickwinkels erkannt. Dann werden beide Arme auf den Boden abgesenkt, um mehr Auflagefläche zu erhalten (Abbildung 4.10(e)). Der Roboter fährt dann die Rampe herauf (Abbildung 4.10(f)) bis zum Ende der Rampe (Abbildung 4.10(g)), woraufhin die Arme wieder in die Fahrposition gebracht werden (Abbildung 4.10(h)).

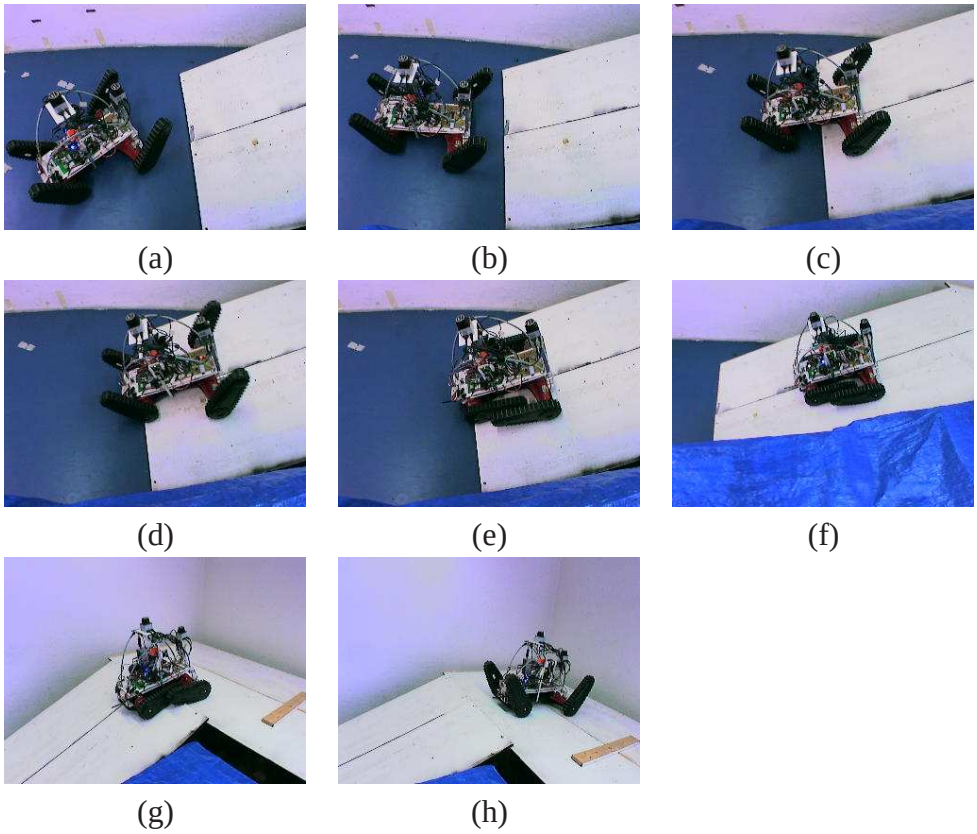
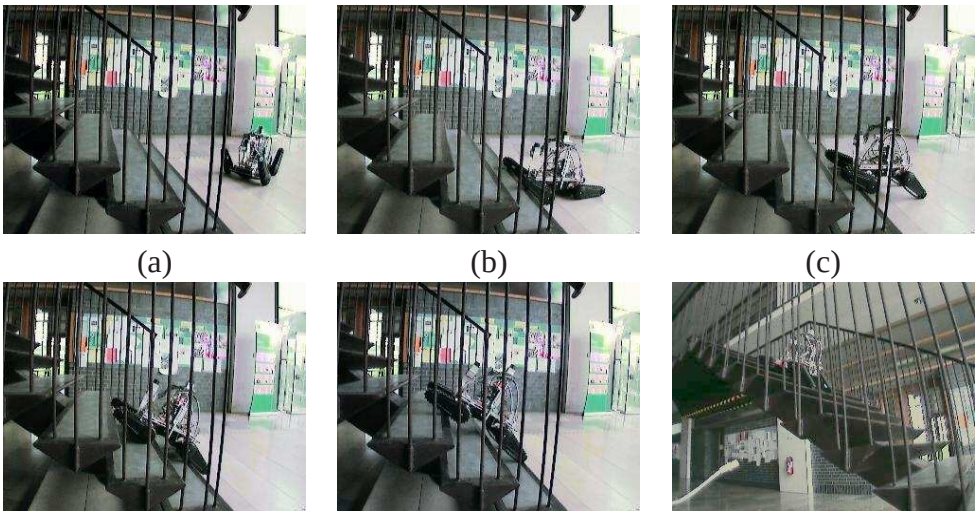


Abbildung 4.10. Rampenfahrt.

Herauffahren einer Treppe Der Roboter startet wie bei der Palettenauffahrt grob ausgerichtet auf eine Treppe (Abbildung 4.11(a)) und richtet sich auf diese aus (Abbildung 4.11(b)). Das Heraufheben auf die erste Stufe (Abbildung 4.11(c)) funktioniert ebenso wie bei der Palettenauffahrt. Danach beginnt der Roboter durch Vorwärtsfahren die Treppe zu erklimmen. Dabei werden die Kontaktsensoren der vorderen Arme benutzt, um festzustellen wo diese auf der Stufe aufliegen und dementsprechend der Armwinkel eingestellt (Abbildung 4.11(d) und (e)). Der Winkel der hinteren Arme wird während der Auffahrt so eingestellt, dass der Roboternickwinkel unabhängig vom Winkel der vorderen Arme konstant bleibt (Abbildung 4.11(f)).



(a) (b) (c)

(d) (e) (f)

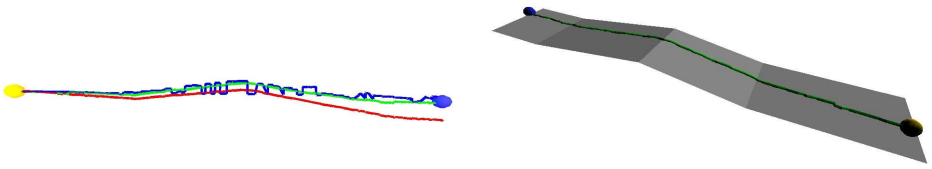
Abbildung 4.11. Herauffahren einer Treppe.

Kapitel 5

Experimentelle Ergebnisse

5.1 Weltmodellierung

Voraussetzung für die Weltmodellierung ist die 3D-Posenschätzung und dabei insbesondere die Höhenschätzung, da die vorhandene erprobte 2D-Lokalisierung diese nicht bereitstellen kann. Ein Beispiel der Höhenschätzung zeigt Abbildung 5.1. Dabei sind die Trajektorien des Roboters, die von der Höhenschätzung generiert wurden, bei der Überfahrt einer Knickrampe zu sehen. Zum Vergleich ist die Grundwahrheit, die manuell nachgemessen wurde, eingeblendet. Gut zur erkennen sind hier die starken Änderungen der Karten-Trajektorie, die durch uninitialisierte Zellen zustande kommen. Aufgrund deren hoher Varianz, können diese Unstetigkeiten durch den Kalman-Filter zuverlässig ausgeglichen werden. Der leichte Driftfehler der Lagesensor-Trajektorie wurde mit dem Kalman-Filter durch die Kartenmessungen kompensiert, so dass eine Trajektorie zustande kommt, die der echten Roboterbewegung durch die Welt entspricht. In einem weiteren Testlauf wurden die Höhen-



(a) (b)

Abbildung 5.1. Höhenschätzung des Roboters im Vergleich zur Grundwahrheit (grau). Zu sehen sind die drei Trajektorien, die die Höhenschätzung nur aufgrund des Lagesensors (rot), nur aufgrund der Karte (blau) und durch den Kalman-Filter verschmolzen (grün) zeigen.

informationen aus dem Weltmodell entnommen und in Abbildung 5.2 aufgetragen. Dabei fuhr der Roboter zuerst eine Strecke von $22m$ auf flachem Boden, so dass als Grundwahrheit eine Höhe von $0m$ angenommen werden konnte. Am Ende war

eine zwei Meter lange Rampe zu bewältigen, deren Höhe nachgemessen und zur Bestimmung der Grundwahrheit im Bereich der Auffahrt linear interpoliert wurde. Abbildung 5.2 zeigt den Fehler über Zeit im Vergleich zu der Höhenschätzung. Da der Roboter zuerst keinen Höhenunterschied zu überwinden hatte, lässt sich dem ersten Teil entnehmen, wie stark die Höhenschätzung über die Strecke driftet. Es zeigte sich, dass der Driftfehler sehr klein bleibt und so in der Karte keine großen Fehler zustande kommen. Erwartungsgemäß ist dann der Fehler während der Rampenauffahrt höher als bei reiner Bodenfahrt. Ein wichtiger Punkt im Hinblick auf die Verwendung der Höhenschätzung zur Weltmodellierung ist die Stetigkeit der Kurve, die keine Sprünge, wie die reine Höhenmessung aufgrund der Karte, vorweist und somit dem Verlauf der echten Höhe entspricht. Dies ist insbesondere wichtig, da Sprünge in der Höhenschätzung zu Sprüngen in der Pose und damit zu Fehlern in der Höhenkarte führen. Tabelle 5.1 gibt die Fehlerwerte während des gesamten Testlaufs, sowie aufgetrennt in den Bereich vor und während der Rampe, an.

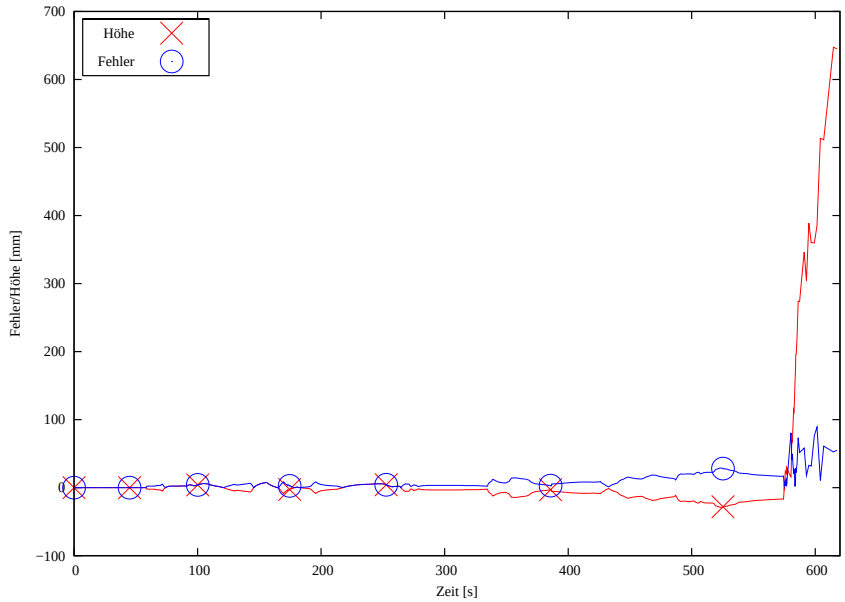


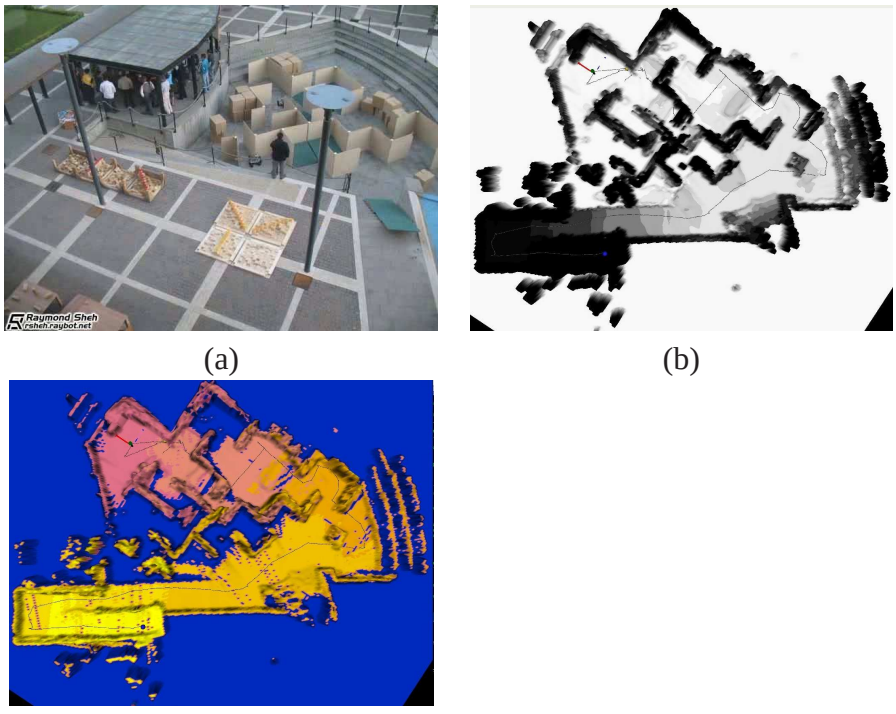
Abbildung 5.2. Fehler der Höhenschätzung während eines Testlaufs. Ab 575 Sekunden begann die Rampenfahrt, was an der starken Steigung der Höhe zu erkennen ist. Der Fehler bei Bodenfahrt ist, wie es zu erwarten ist, deutlich geringer.

Zur Evaluation der Weltmodellierung soll hier eine Höhenkarte einer Testarena, die vom NIST [11] während des Rescue Robotics Camp 2006 in Rom aufgebaut wurde, dienen. In Abbildung 5.3(a) ist dieses Testareal zu sehen, wobei Abbildung 5.3(b) die dort gebaute Höhenkarte zeigt. Dabei stehen dunklere Gebiete für höher gelegene Regionen. Die Teststrecke führte vom tiefer gelegenen Areal über

	Mittlerer Fehler	Mittlerer Quadratischer Fehler
Vor der Rampe	5.49mm	8.45mm
Auf der Rampe	36.58mm	43.63mm
Gesamt	6.00mm	10.09mm

Tabelle 5.1. Fehler der Höhenschätzung.

mehrere kleine Rampen zu der langen Rampe, die in Abbildung 5.4 oben zu sehen ist, und schließlich diese hinauf. Die Varianzkarte in Abbildung 5.3(c) zeigt gut, wie sich die Varianz mit zunehmender Entfernung von der aktuellen Roboterposition am Ende der Rampe erhöht (siehe Abschnitt 3.3). Dazu wurde der Varianzpropagationsschritt für alle Zellen einmal von Hand ausgelöst. In Abbildung 5.4 ist noch einmal die gesamte Karte in einer dreidimensionalen Perspektive zu sehen. Darauf sind die realen Strukturen, insbesondere die Rampe am Ende und die äußeren Treppen, gut zu erkennen. Die erstellte Höhenkarte stellt somit eine gute Repräsentation der echten Umgebung dar.



(a) (b)

(c)

Abbildung 5.3. Höhenkarten vom Rescue Robotics Camp 2006: (a) Die Testarena, die vom NIST aufgebaut wurde. (b) Die entsprechende Höhenkarte, die sich von weiß (niedrige Höhe) nach schwarz (große Höhe) erstreckt. (c) Die Varianzen der Höhenzellen, von rosa (hohe Varianz) nach gelb (niedrige Varianz).

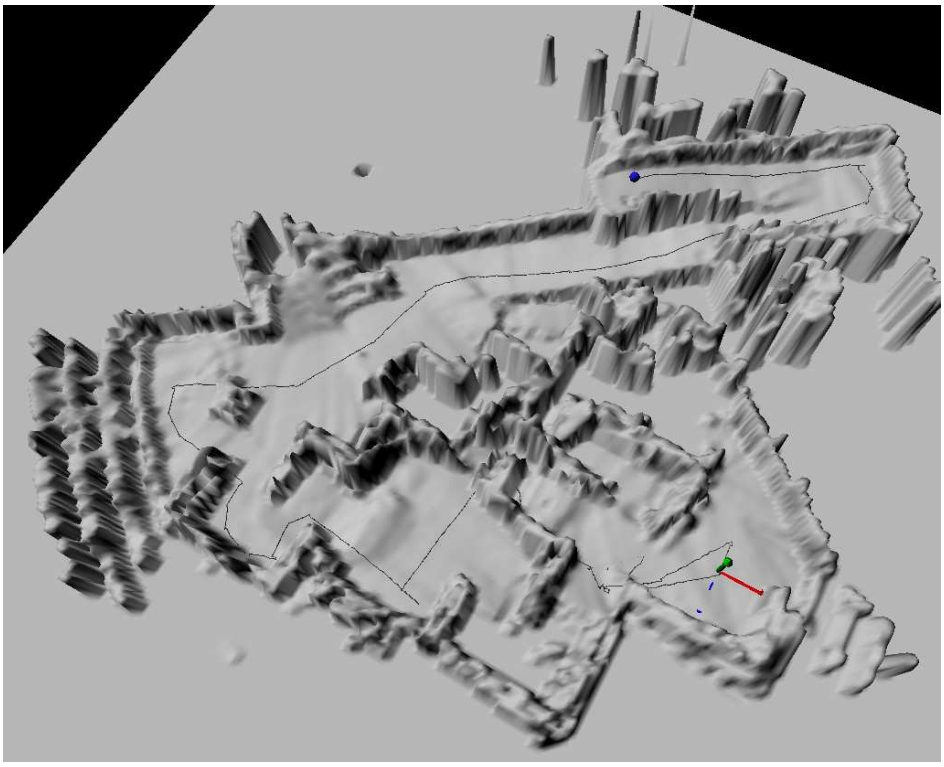


Abbildung 5.4. Dreidimensionale Ansicht der Höhenkarte vom Rescue Robotics Camp 2006.

5.2 Planung auf Verhaltenskarten

5.2.1 Klassifikation

Zur Evaluation der Klassifikationsverfahren wurde die Vorklassifikation und die MRF-Klassifikation mit handklassifizierten Daten verglichen. Abbildung 5.5 zeigt eine Beispielkarte, auf die sowohl die Vorklassifikation als auch die Klassifikation mit Markov-Random-Fields angewandt wurde, im Vergleich zur idealen Klassifikation, die manuell erzeugt wurde. Es ist gut zu erkennen, dass die Klassifikation beider Verfahren sehr nahe an der von Hand vorgenommenen Klassifikation liegt und somit eine sehr gute Basis für die nachfolgenden Planungsalgorithmen liefert, wenn man davon absieht, dass die Vorklassifkation keine Rampe erkennen kann. Insbesondere entspricht die Unterscheidung zwischen Boden und Wand der Vorgabe durch die Grundwahrheit. Unterschiede, die innerhalb des befahrbaren Bereichs liegen und damit für den Roboter relevant sind, sind nur am Ende der Rampe zu erkennen, wo die MRF-Klassifikation einen Teil der Rampe der Hindernisklasse zugeordnet hat. Die kann allerdings durch das an dieser Stelle nicht perfekte Weltmodell verursacht sein.

Eine quantitative Möglichkeit zur Evaluation einer Klassifikation bieten Konfusionsmatrizen, die zu einem Klassifikationsverfahren die Anzahl der Klassenzuordnungen nach den einzelnen Klassen aufschlüsseln. Für die Vorklassifikation ist die Konfusionsmatrix in Tabelle 5.2 und für die MRF-Klassifikation in Tabelle 5.3 angegeben. Diesen lässt sich auch die grundsätzliche Information der richtig klassifizierten Zellen entnehmen, wenn man die unentscheidbaren Zellen der Vorklassifikation als Hinderniszellen wertet. Es ergeben sich für die Vorklassifikation 81.46% und für die MRF-Klassifikation 86.70% korrekt klassifizierter Zellen. Dadurch lässt sich grundsätzlich schon erkennen, dass das MRF-Verfahren erwartungsgemäß besser ist. Allerdings ist der reine Prozentsatz an richtig klassifizierten Zellen kein ausreichendes Maß zur Bewertung der Klassifikationsqualität. Dies liegt zum einen daran, dass beide Verfahren sehr viele triviale Klassifikationen vorzunehmen hatten, die also keine wirkliche Aussage über die Qualität des Algorithmus zulassen. So ist zum Beispiel eine Zelle, deren Nachbarzelle mehr als $40cm$ höher liegt, natürlich sehr einfach als Wand zu erkennen.

Zum anderen sind manche Fehlzuordnungen durch die Klassifikation für die Planungsalgorithmen sehr viel kritischer als andere. Die Klassifikation einer Wandzelle als Hinderniszelle, kann relativ unbedeutend sein, wenn diese von als Wand klassifizierten Zellen umgeben ist, da somit kein Plan möglich ist. Die Klassifkation einer einzigen Bodenzelle als Wandzelle ist allerdings, selbst wenn diese von Bodenzellen umgeben ist, fatal, da so eigentlich befahrbare Wege als untraversierbar erkannt werden. Gut zu sehen ist, dass beide Verfahren die Unterscheidung von Boden und

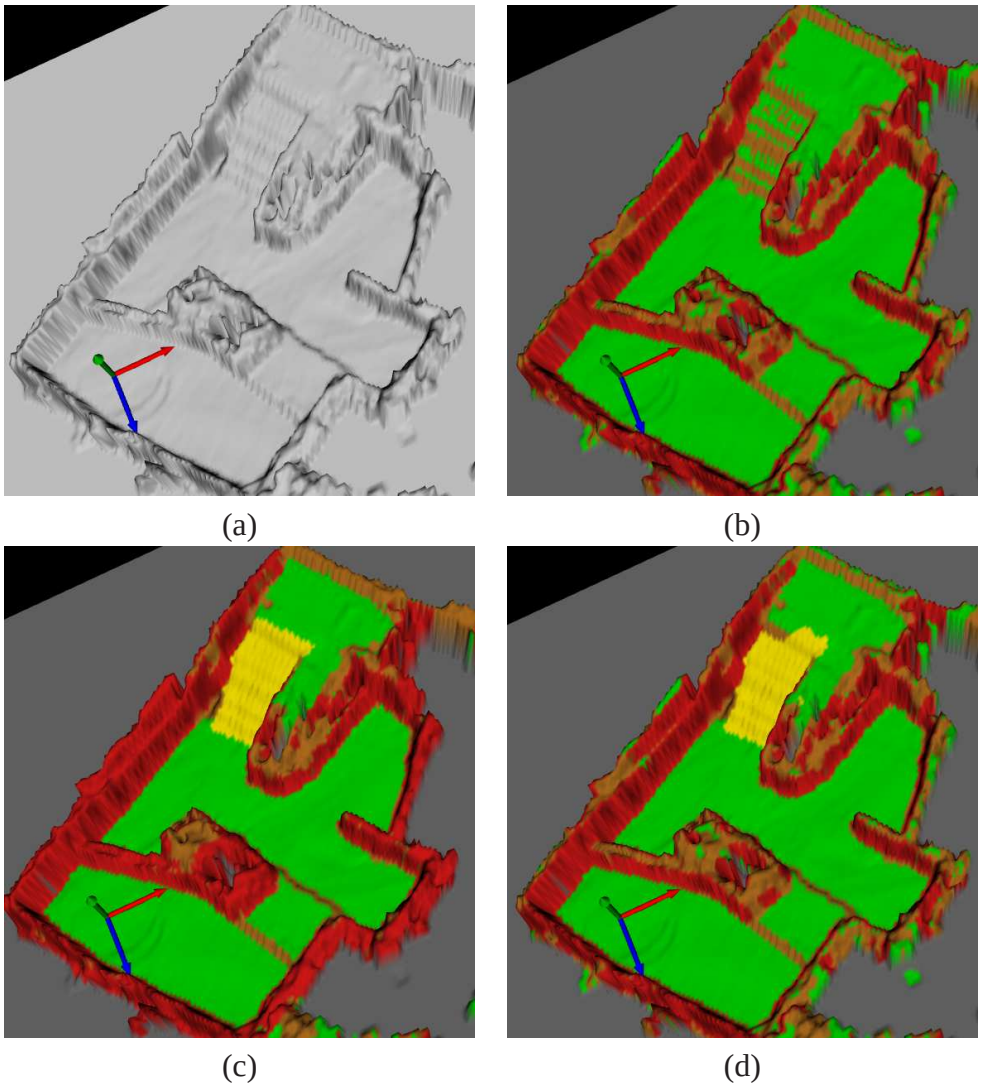


Abbildung 5.5. Vergleich der Vorklassifikation und der MRF-Klassifikation mit handklassifizierten Daten. Dabei sind Bodenzellen grün und Wandzellen rot eingefärbt. Braune Zellen markieren Hindernisse, beziehungsweise bei der Vorklassifikation nicht entscheidbare Zellen und gelbe Zellen eine Rampe. (a) Unklassifizierte Höhenkarte. (b) Vorklassifikation. (c) Von Hand klassifizierte Daten. (d) MRF-Klassifikation.

Wand grundsätzlich sehr zuverlässig beherrschen und insbesondere die Fehlklassifizierung einer Bodenzelle als Wand- oder Hindernis äußerst selten vorkommt (Vorklassifikation: 36/6336, MRF-Klassifikation: 23/6336). Beim umgekehrten Fall der als Boden fehlklassifizierten Zellen, stammen diese häufig auch daher, dass sie nahe an Wänden oder Hindernissen liegen, was natürlich eine Fehlzuordnung begünstigt, allerdings in der Praxis relativ unproblematisch ist. Die Konfusionsmatrizen zeigen auch, dass die meisten Fehlklassifikationen von der Unterscheidung zwischen Hindernis und Wand herrühren. Beide Verfahren ordnen relativ häufig Wandzellen die Hindernisklasse zu. Wie auch in den Beispielen in Abbildung 5.5(b) und (d) zu erkennen ist, liegen diese meistens in der Nähe von Wandzellen, so dass diese für die Anwendung auch keinerlei Problem darstellen. Der auffälligste Unterschied zwischen beiden Verfahren ist, dass die Vorklassifikation keine Rampen erkennt, was allerdings auch nicht vorgesehen ist. Zur Erkennung komplexer Strukturen sollen ja gerade die Markov-Random-Fields zu Einsatz kommen. Da die Vorklassifikation lediglich als Echtzeitverfahren zur Detektion von einfach zu befahrenen Terrain dienen soll, kann auch nur die Unterscheidung zwischen Boden und nicht-Boden als Bewertungsmaßstab heranziehen, die in 91.22% der Fälle korrekt ist. Die Klassifikation mittels Markov-Random-Fields leistet diese Unterscheidung ebenso besser, und zwar in 96.81% der Fälle.

Abschließend lässt sich sagen, dass beide Verfahren eine sehr gute Unterscheidung von traversierbaren und nicht-traversierbaren Terrain, insbesondere auch im Hinblick auf die Verwendung zur autonomen Navigation, liefern. Das Unvermögen der Vorklassifikation zur Erkennung von komplexeren Strukturen, wie zum Beispiel Rampen, wird durch die Verwendung der Markov-Random-Fields auf den unerkannten Gebieten zuverlässig ausgeglichen. Dennoch ist die Vorklassifikation ein wichtiger Bestandteil des Gesamtsystems, da diese in Echtzeit laufen kann, und somit verhindert, dass der Roboter auf den Abschluss einer MRF-Klassifikation warten muss, bevor er weiter fahren kann. Experimentell wurden für die Vorklassifikation eine Laufzeit von $32.27ms \pm 2.48ms$ auf einer Karte von 60.000 Zellen auf einem AMD Athlon X2 3800+ ermittelt und für die MRF-Klassifikation eine Laufzeit von $26.31s \pm 27.88s$. Die hohe Standardabweichung der MRF-basierten Klassifikation kommt daher, dass die Laufzeit der MRF-Inferenz nicht linear mit der Größe des MRF-Graphen zusammenhängt. Empirische Beobachtungen zeigten eine quadratische Laufzeit, die somit bei verschieden großen zu klassifizierenden Gebieten, stark schwanken kann. Die MRF-Klassifikation läuft deshalb während der Exploration im Hintergrund und ermöglicht es so dem Roboter komplexere Hindernisse zu erkennen, ohne die Fahrt zu unterbrechen. Die größeren Laufzeiten der MRF-Klassifikation stellten allerdings in der Praxis kein Problem da, da der relativ unwahrscheinliche Fall, dass der Roboter das gesamte Gebiet exploriert hat, bevor die

MRF-Klassifikation abgeschlossen ist und dieser somit auf die MRF-Klassifikation warten muss, nie eingetreten ist.

Vorklassifikation \ echte Klasse	Boden	Rampe	Hindernis	Wand
Boden	6300	261	183	325
Unentscheidbar	26	331	1560	1109
Wand	10	17	174	2845

Tabelle 5.2. Konfusionsmatrix der Vorklassifikation.

MRF \ echte Klasse	Boden	Rampe	Hindernis	Wand
Boden	6313	11	123	270
Rampe	22	574	0	0
Hindernis	1	24	1675	1178
Wand	0	0	118	2831

Tabelle 5.3. Konfusionsmatrix der MRF-Klassifikation.

5.2.2 Gesamtsystem

Zur Evaluation des gesamten Systems wurde ein vollständig autonomer Testlauf durchgeführt, bei dem der Roboter eine Höhenkarte der Umgebung erstellen, sowie die Hindernisse darauf erkennen und überwinden musste. Der verwendete Testparcours ist in Abbildung 5.6 zu sehen. Um diesen vollständig zu explorieren muss der Roboter die Fähigkeit besitzen autonom Rampen und Paletten zu erkennen und zu überqueren. Da in den vorherigen Abschnitten schon auf die Weltmodellierung ein-

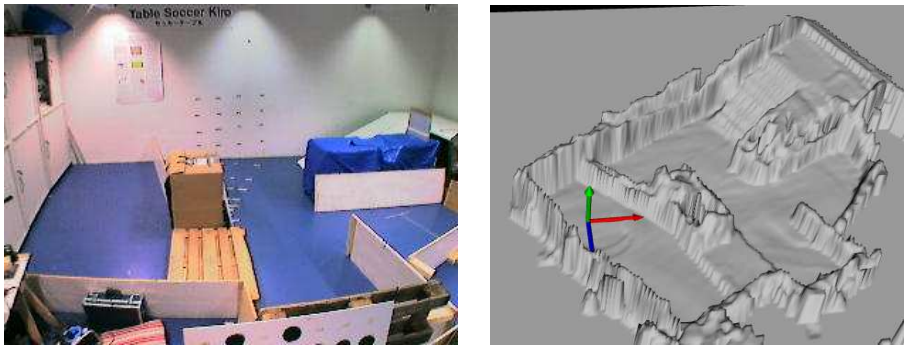


Abbildung 5.6. (a) Der Testparcours für den autonomen Lauf. Der Roboter startet im Koordinatenursprung, der sich im linken Bereich befindet, welcher durch eine Palette vom Mittelteil abgetrennt ist. Von diesem aus gibt es eine Rampe, die die Möglichkeit bietet das kleine Plateau rechts oben zu erreichen. (b) Die Höhenkarte der Testarena, die der Roboter während des Testlaufs gebaut hat.

gegangen wurde, soll hierbei der Fokus auf der Planung und der Verhaltensauswahl liegen. Da die benutzte A*-Planung per Definition optimal ist und auch im Anwendungsbereich der Roboterexploration in zweidimensionalen Umgebungen, die zum Beispiel durch Occupancy-Grids modelliert sind, schon vielfältig untersucht wurde, sollen hier insbesondere die Stellen beachtet werden, an denen Hindernisse vorkommen und damit die Notwendigkeit besteht, dass die Hindernisse, zu denen Fähigkeitsbeschreibungen vorhanden sind, erkannt werden und der Roboter der *Verhaltenskarte* entsprechend richtig handelt. Deshalb sollen insbesondere Entscheidungssituationen in der Nähe von *Übergangskanten* dargestellt werden, um einerseits sicherzustellen, dass der Roboter, wie in Abschnitt 4.2 beschrieben, die Exploration auf flachem Grund dem Hindernisüberfahren vorzieht, wenn er die Möglichkeit dazu hat. Andererseits soll gezeigt werden, dass der Roboter, falls ein Hindernis überwunden werden muss, auch die Fähigkeit besitzt über dieses zu planen und die jeweils passenden Verhalten zuverlässig auszuführen.

Dazu werden in Abbildung 5.7 vier interessante Zeitpunkte des kompletten Laufs herausgegriffen. Die erste aufschlussreiche Situation findet sich, wenn der Roboter, wie in (a),(b) und (c) zu sehen während der Exploration auf die Palette trifft. In der

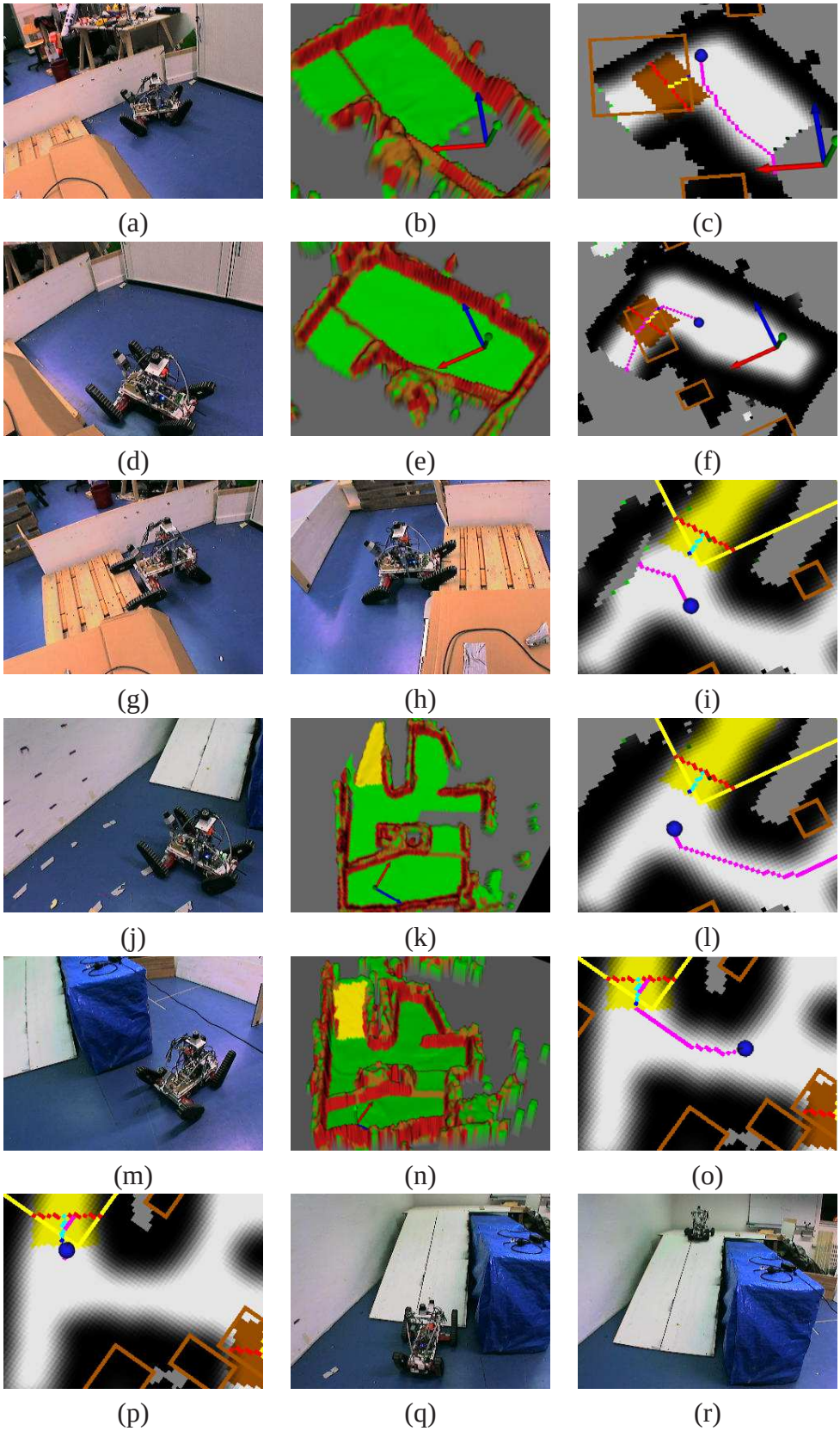


Abbildung 5.7. Ausgewählte Situationen eines autonomen Testlaufs.

Höhenkarte in (b) ist deutlich ein noch unbekannter Bereich (grau) zu sehen und die Verhaltenskarte in (c) zeigt die erkannte Palette, sowie in hellgrün und damit mit hohem Nutzen Frontier-Zellen, die hinter der Palette liegen. Dennoch entscheidet sich der Roboter zuerst ein Ziel im unexplorierten Bereich anzufahren, da die Kosten der Hindernisüberfahrt zu hoch sind, und plant zu einer Frontier-Zelle in diesem Bereich (violett).

In der Höhenkarte in (e) ist zu erkennen, dass der Roboter den Bereich vor der Palette nun vollständig erkundet hat. Dies führt dazu, dass nur noch Ziele mit sehr hohen Kosten vorhanden sind. Der Roboter kann also, wie in (f) zu erkennen, einen Pfad über die Palette planen. Bei Erreichen des braunen Bereichs in der Verhaltenskarte, der die Verhaltensroutine *Palette* anzeigt, stellt der Roboter zuerst die Vorabbedingungen sicher und führt dann wie geplant das entsprechende Verhalten aus (zu sehen in (g) und (h)). Die Exploration im mittleren Bereich führt den Roboter, wie man in (j) erkennen kann, in die Nähe der Rampe, wo es noch uninitialisierte Zellen gibt, die in der Verhaltenskarte in (i) zu erkennen sind. Als der Roboter diese exploriert hat, entscheidet er sich, obwohl er, wie in (l) gut zu sehen ist, die Rampe detektiert hat, den Bereich direkt hinter sich weiter zu erforschen. Wie schon vor der Palette verhindern die hohen Kosten für die Rampenfahrt, dass der Roboter diese vorzieht, wenn es, wie die Höhenkarte in (k) zeigt, noch unexplorierte Bereiche gibt, die einfacher zu erreichen sind.

Die Karte in (n) zeigt, dass der Roboter schließlich den kompletten Boden erkundet hat und nun in (o) einen Pfad, der auf die Rampe führt, plant. In (p) und (q) kann man sehen, dass der Roboter vor der Rampenfahrt zuerst die Vorabbedingungen und hier insbesondere die Startrichtung, die in der Verhaltenskarte in cyan markiert ist, sichergestellt hat. Am Ende fährt der Roboter in (r) die Rampe herauf und hat damit die komplette Testarena exploriert.

Ein Video, das unter anderem diesen Testlauf mit den Entscheidungssituationen zeigt, ist unter `www.informatik.uni-freiburg.de/~rescue/robots/video/behaviour_maps.mpg` erhältlich.

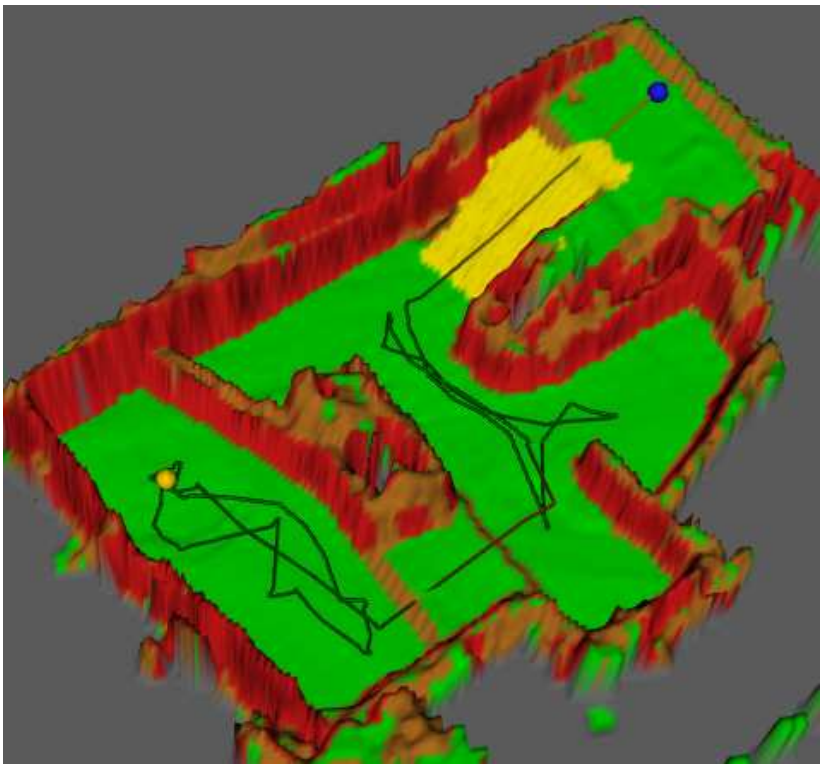
Die vollständige Karte, die der Roboter während des Testlaufs erstellt hat, mit der Trajektorie der Fahrt ist in Abbildung 5.8(a) zu sehen. Dabei wurde eine Strecke von 23.00m Länge gefahren, was 9 Minuten und 55 Sekunden dauerte. Die Höhenkarte stellt eine gute Repräsentation der realen Testumgebung dar und die Klassifikation zeigt eine zuverlässige Erkennung der spezifischen Hindernisklassen. Die Verhaltenskarte in Abbildung 5.8(b) zeigt, dass alle *Übergangskanten* zu Hindernissen, von denen der Roboter eine passende Fähigkeitsbeschreibung besaß, zuverlässig erkannt wurden. Da der Roboter aufgrunddessen dieses Areal vollständig autonom exploriert hat, ist auch sichergestellt, dass die entsprechenden Vorabbedingungen, die berechnet wurden, für die Ausführung der jeweils zugehörigen Hindernisroutine sehr gut geeignet waren. Die in Abbildung 5.8(c) zu sehende Kostenfunktion

zeigt, wie es dem Roboter möglich war, ein zu häufiges Überfahren von Hindernissen durch Beachten der hohen Kosten zu vermeiden. Dies wird insbesondere durch die Trajektorie in Abbildung 5.8(a) deutlich, die zeigt, wie der Roboter jeden Bereich erst vollständig exploriert, bevor er ein Hindernis überquert. Das führt dazu, dass jedes Hindernis nur einmal überwunden werden muss, und der Roboter somit hinsichtlich der Fehlervermeidung die optimale Trajektorie gefahren ist.

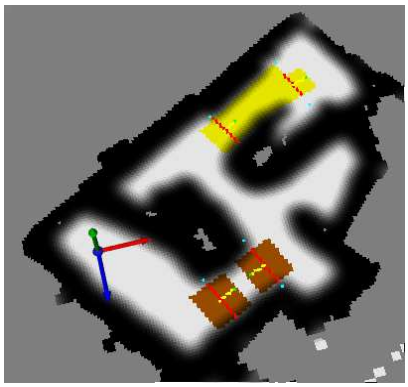
Um die Echtzeitfähigkeiten des Systems zu zeigen, listet Tabelle 5.4 die Laufzeiten der verschiedenen Komponenten auf einem AMD Athlon X2 3800+ auf. Die Werte für die Berechnung der Vorabbedingungen, der Kartenvorverarbeitung, der Kartenfilterung und der Vorklassifikation wurden dabei auf der vollständigen und damit auch größtmöglichen Karte des Testlaufes mit 60.000 Zellen ermittelt, um die Ergebnisse nicht durch die anfänglich sehr kleinen Karten zu beschleunigen und damit zu verfälschen. Die Werte zur Scan-Integration, Posenschätzung, Planung und MRF-Klassifikation wurden während aller Zyklen eines Testlaufs aufgenommen, da diese nicht direkt von der Kartengröße abhängig sind. Die Rechenzeit bei der Berechnung der Vorabbedingungen kommt durch die Hough-Transformation zustande und die der Kartenvorverarbeitung wird hauptsächlich durch das Weichzeichnen der Karte mit einem Gauss-Kern verursacht. Beide Verfahren, sowie auch die Vorklassifikation und Kartenfilterung arbeiten in der aktuellen Implementierung auf der gesamten Karte. Dies ist sicherlich noch nicht optimal, da diese nur auf den durch den letzten Scan geänderten Zellen und deren Nachbarn neue Informationen berechnen. Eine verbesserte Implementierung kann hier noch einmal deutliche Senkungen der Laufzeiten bringen. Dennoch lassen alle Berechnungen schon jetzt eine problemlose Ausführung in Echtzeit zu. Die einzige Ausnahme bildet die MRF-Inferenz, die allerdings wie schon in Abschnitt 5.2.1 beschrieben im Hintergrund läuft.

Verfahren	Rechenzeit	Standardabweichung
Scan Integration (683 Strahlen)	$1.88ms$	$0.47ms$
3D-Posenschätzung	$0.09ms$	$0.01ms$
Kartenfilterung	$55.03ms$	$3.96ms$
Vorklassifikation	$32.27ms$	$2.48ms$
Berechnung von Vorabbedingungen	$63.94ms$	$0.78ms$
Kartenvorverarbeitung	$37.42ms$	$0.98ms$
A*-Planung	$32.26ms$	$4.76ms$
MRF-Klassifikation	$26.31s$	$27.88s$

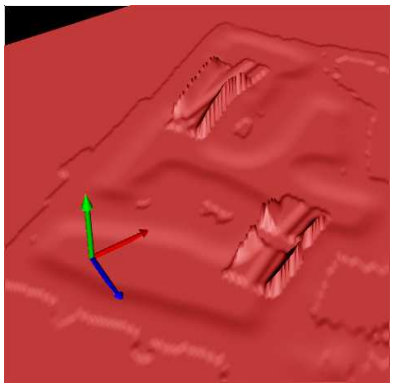
Tabelle 5.4. Rechenzeiten der verschiedenen Komponenten.



(a)



(b)



(c)

Abbildung 5.8. Ergebnisse der Exploration. (a) Die klassifizierte Höhenkarte mit der Trajektorie des Roboters. (b) Die Verhaltenskarte, die aus der Höhenkarte erzeugt wurde. Die *Übergangskanten* auf die Palette und auf die Rampe sind rot markiert. (c) Eine Visualisierung der Kostenfunktion der Verhaltenskarte. Gut zu erkennen sind die hohen Kosten bei den Hindernisübergängen, die eine Planung über diese verhindern, wenn es einfachere Pfade gibt. Dies führt zu einer Explorationstrajektorie, die jedes Hindernis nur einmal überquert.

Kapitel 6

Fazit

In dieser Diplomarbeit wurde ein System entwickelt, dass es einem autonomen Roboter ermöglicht, schwer zugängliches Terrain zu klassifizieren und auf diesem mit Einbezug der eventuell auftretenden Hindernisse zu planen. Dazu war es nötig während der Exploration eine $2\frac{1}{2}$-dimensionale Weltmodellierung vorzunehmen, was durch die Konstruktion von Höhenkarten mit einem geneigten Laserscanner erfolgte. Diese wurden dann mit Markov-Random-Fields klassifiziert, um komplexe Hindernisse zu erkennen. Um die Navigation auf schwerem Terrain zu ermöglichen, war es nötig, dass der Roboter unter Beachtung der spezifischen Hindernisse Pfade über diese planen konnte. Aufbauend auf einer allgemeinen Formulierung zur Integration von verschiedenen Fertigkeiten durch *Fähigkeitsbeschreibungen*, wurden dazu *Verhaltenskarten* entwickelt, die es dem Roboter erlauben, aufgrund der Planung kontextspezifische Verhalten auszuführen.

Mittels verschiedener Fähigkeitsroutinen konnte so ein System entwickelt werden, das in Echtzeit Gebiete mit Hindernissen, wie zum Beispiel Rampen und Paletten, kartographieren, klassifizieren und explorieren kann. Alle Algorithmen wurden auf einem echten Roboter implementiert und die Möglichkeiten des Systems wurden schließlich durch einen vollständig autonomen Testlauf unter Beweis gestellt.

Weiterführende Arbeiten an dem System werden sich zum einen mit der Verbesserung der Weltmodellierung und zum anderen mit der Erweiterung durch neue Verhaltensroutinen beschäftigen. Um die Qualität der Karten zu erhöhen scheint es sinnvoll eine Rückmeldung zur Verbesserung der Pose nicht nur aus dem 2D-SLAM Algorithmus und aus der Karte durch eine getrennte Höhenschätzung zu entnehmen, sondern jeweils die gesamten sechs Freiheitsgrade der Pose mit Erhalt eines Scans zu verbessern. Des weiteren lässt sich die Ausdruckskraft der Karten durch die Verwendung von Multi-Level-Surface-Maps verbessern, ohne dass der Speicherverbrauch oder Rechenzeit unverhältnismäßig stark wachsen, da sämtliche benutzten Algorithmen ohne Probleme darauf übertragbar sind. Außerdem ist die Weltrepräsentation als eine einzige zellenbasierte Karte sehr ungünstig, da diese relativ schlecht mit zu-

nehmender Größe skaliert. Deshalb wird hier eine komplexere Repräsentation zur Anwendung kommen, die mehrere Teilkarten verwalten kann und so dieses Problem löst.

Bei der Integration neuer Verhalten liegt durch die Verwendung von Fähigkeitsbeschreibungen die Hauptarbeit nur noch in der Erzeugung der eigentlichen Verhaltensroutine. Die aktuellen Verhalten befähigen den Roboter autonom Paletten herauf- und herunter zu fahren, Rampen zu fahren und Treppen heraufzufahren. So ist es zum Beispiel relativ einfach denkbar ein Verhalten zum Herauffahren einer Rampe, die statt auf dem Boden auf einer Erhöhung wie einer Palette beginnt, zu implementieren. Grundsätzlich ist das hier beschriebene System nicht nur auf das Ausführen von Hindernisroutinen beschränkt. So ist es zum Beispiel denkbar, unter der Voraussetzung, dass eine Tür erkannt werden kann, einen Pfad zu dieser Tür zu planen, wobei das Verhalten, das den Roboter durch eine Tür steuert, eine geeignete Routine besitzt, die automatisch ausgeführt wird und mittels eines Manipulators diese autonom öffnen kann. Solch eine, für Menschen, simple Fähigkeit in ein vollständig autonomes Robotiksystem zu integrieren erscheint gerade deshalb sinnvoll, weil Roboter zunehmend auch in menschlichen Umgebungen eingesetzt werden, in denen Aktionen wie das Öffnen von Türen und das Treppenfahren eine Selbstverständlichkeit sind, die ein Roboter leisten können muss.

Literaturverzeichnis

[1] ALLEN, James F.: Maintaining knowledge about temporal intervals. In: *Commun. ACM* 26 (1983), Nr. 11, S. 832–843. – ISSN 0001–0782

[2] ANGUELOV, D. ; TASKAR, B. ; CHATALBASHEV, V. ; KOLLER, D. ; GUPTA, D. ; HEITZ, G. ; NG, A.Y.: Discriminative Learning of Markov Random Fields for Segmentation of 3D Range Data. In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. San Diego, California, June 2005

[3] BARES, J. ; HEBERT, M. ; KANADE, T. ; KROTKOV, E. ; MITCHELL, T. ; SIMMONS, R. ; WHITTAKER, W. R. L.: Ambler: An autonomous rover for planetary exploration. In: *IEEE Computer Society Press* 22 (1989), Nr. 6, S. 18–22

[4] BRUCE, J. ; BALCH, T. ; VELOSO, M.: Fast and Inexpensive Color Image Segmentation for Interactive Robots. In: *Proceedings of IROS-2000*. Japan, October 2000

[5] DISSANAYAKE, M. ; NEWMAN, P. ; CLARK, S. ; WHYTE, Durrant H. F. ; CSORBA, M.: A solution to the simultaneous localization and map building (SLAM) problem. In: *IEEE Transactions on Robotics and Automation* 17 (2001), Nr. 3, S. 229–241

[6] DORNHEGE, Christian ; KLEINER, Alexander: Visual Odometry for Tracked Vehicles. In: *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR '06)*, 2006

[7] ELFES, A.: Using Occupancy Grids for Mobile Robot Perception and Navigation. In: *Computer* 22 (1989), Nr. 6, S. 46–57. – ISSN 0018–9162

[8] HÄHNEL, D.: *Mapping with Mobile Robots*. Freiburg, Deutschland, Universität Freiburg, Dissertation, 2005

[9] HEBERT, M. ; MACLACHLAN, R. ; CHANG, P.: Experiments with Driving Modes for Urban Robots. In: *Proceedings of SPIE*, 1999

[10] HOUGH, P.V.C.: Methods and Means for Recognizing Complex Patterns. In: *U.S. Patent 069654*, 1962

[11] JACOFF, A. ; MESSINA, E. ; EVANS, J.: Experiences in Deploying Test Arenas for Autonomous Mobile Robots. In: *Proc. of the PerMIS Workshop*. Mexico, 2001

[12] KALMAN, R. E.: A New Approach to Linear Filtering and Prediction Problems. In: *Trans. ASME Basic Engineering* 82 (1960), März, Nr. 1, S. 35–45

[13] KLEINER, A. ; DORNHEGE, C. ; KUEMMERLE, R. ; RUHNKE, M. ; B.STEDER ; NEBEL, B. ; DOHERTY, P. ; WZOREK, M. ; RUDOL, P. ; CONTE, G. ; DURANTE, S. ; LUNDSTROM, D.: RoboCupRescue - Robot League Team RescueRobots Freiburg (Germany). In: *Robocup 2006, (CDROM Proceedings)*. Bremen, Germany, 2006. – Winner of the autonomy competition

[14] KLEINER, A. ; STEDER, B. ; DORNHEGE, C. ; HOEFLER, D. ; MEYER-DELIUS, D. ; PREDIGER, J. ; STUECKLER, J. ; GLOGOWSKI, K. ; THURNER, M. ; LUBER, M. ; SCHNELL, M. ; KUEMMERLE, R. ; BURK, T. ; BRAEUER, T. ; NEBEL, B.: RoboCupRescue - Robot League Team RescueRobots Freiburg (Germany), Team Description Paper. In: *Rescue Robot League, (CDROM Proceedings)*. Osaka, Japan, 2005. – Winner of the autonomy competition

[15] KÜMMERLE, Rainer: *Strukturerkennung in 3D-Daten durch Markov Random Fields*. Freiburg, Deutschland, Universität Freiburg, Studienarbeit, 2006

[16] LEONARD, J.J. ; DURRANT-WHYTE, H.F.: Mobile robot localization by tracking geometric beacons. In: *IEEE Transactions on Robotics and Automation* 7 (1991), Nr. 4, S. 376–382

[17] LUCAS, B.D. ; KANADE, T.: An Iterative Image Registration Technique with an Application to Stereo Vision. In: *IJCAI81*, 1981, S. 674–679

[18] MANDUCHI, R. ; CASTANO, A. ; TALUKDER, A. ; MATTHIES, Larry: Obstacle Detection and Terrain Classification for Autonomous Off-Road Navigation. In: *Auton. Robots* 18 (2005), Nr. 1, S. 81–102

[19] MAYBECK, P. S.: *Mathematics in Science and Engineering*. Bd. 141: *Stochastic models, estimation, and control*. Academic Press, 1979

[20] OJEDA, L. ; BORENSTEIN, J. ; WITUS, G.: Terrain trafficability characterization with a mobile robot. In: GERHART, G. R. (Hrsg.) ; SHOEMAKER, C. M. (Hrsg.) ; GAGE, D. W. (Hrsg.): *Unmanned Ground Vehicle Technology VII* Bd. 5804, 2005, S. 235–243

[21] PFAFF, P. ; TRIEBEL, R. ; BURGARD, W.: An Efficient Extension to Elevation Maps for Outdoor Terrain Mapping and Loop Closing. In: *International Journal of Robotics Research (IJRR) - Special Issue of International Conference on Field and Service Robotics (FSR)*, 2005

[22] RUSSELL, Stuart J. ; NORVIG, Peter: *Artificial Intelligence. A Modern Approach*. Prentice-Hall, 1995

[23] SHI, Jianbo ; TOMASI, Carlo: Good Features to Track. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*. Seattle, Juni 1994

[24] SMITH, R. ; SELF, M. ; CHEESEMAN, P.: Estimating Uncertain Spatial Relationships in Robotics. In: LEMMER, J. F. (Hrsg.) ; KANAL, L. N. (Hrsg.): *Uncertainty in Artificial Intelligence 2*. Amsterdam : North-Holland, 1988, S. 435–461

[25] STEDER, Bastian: *Reinforcement-Lernen zur Verhaltensauswahl eines Rettungsroboters*. Freiburg, Deutschland, Universität Freiburg, Studienarbeit, 2006

[26] TALUKDER, A. ; MANDUCHI, R. ; CASTANO, R. ; OWENS, K. ; MATTHIES, L. ; CASTANO, A. ; HOGG, R.: Autonomous terrain characterisation and modelling for dynamic control of unmanned vehicles. In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2002, S. 708–713

[27] TASKAR, B. ; CHATALBASHEV, V. ; KOLLER, D.: Learning Associative Markov Networks. In: *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, 2004

[28] THRUN, S. ; BURGARD, W. ; FOX, D.: A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In: *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000, S. 321–328

[29] THRUN, S. ; MONTEMERLO, M. ; DAHLKAMP, H. ; STAVENS, D. ; ARON, A. ; DIEBEL, J. ; FONG, P. ; GALE, J. ; HALPENNY, M. ; HOFFMANN, G. ; LAU, K. ; OAKLEY, C. ; PALATUCCI, M. ; PRATT, V. ; STANG, P. ; STROHBAND, S. ; DUPONT, C. ; JENDROSSEK, L.-E. ; KOELEN, C. ; MARKEY, C. ; RUMMEL, C. ; NIEKERK, J. van ; JENSEN, E. ; ALESSANDRINI, P. ; BRADSKI, G. ; DAVIES, B. ; ETTINGER, S. ; KAEHLER, A. ; NEFIAN, A. ; MAHONEY, P.: Winning the DARPA Grand Challenge. In: *JFR* (2006). – accepted for publication

[30] TRIEBEL, R. ; PFAFF, P. ; BURGARD, W.: Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing. In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. Beijing, China, 2006

[31] VANDAPEL, N. ; HUBER, D. ; KAPURIA, A. ; HEBERT, M.: Natural Terrain Classification using 3-D Ladar Data. In: *IEEE International Conference on Robotics and Automation* Bd. 5, 2004, S. 5117 – 5122

[32] VIOLA, Paul ; JONES, Michael: Rapid Object Detection using a Boosted Cascade of Simple Features. In: *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, 2001

[33] WALDRON, K.J. ; ARKIN, R.C. ; BAKKUM, D. ; MERRILL, E. ; ABDALLAH, M.: Proprioceptive control for a robotic vehicle over geometric obstacles. In: *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003

[34] WOLF, D.F. ; SUKHATME, G. ; FOX, D. ; BURGARD, W.: Autonomous Terrain Mapping and Classification Using Hidden Markov Models. In: *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005

[35] YAMAUCHI, B.: A Frontier-Based Approach for Autonomous Exploration. In: *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)*, 1997

[36] YE, C. ; BORENSTEIN, J.: A new terrain mapping method for mobile robot obstacle negotiation. In: GERHART, G. R. (Hrsg.) ; SHOEMAKER, C. M. (Hrsg.) ; GAGE, D. W. (Hrsg.): *Unmanned Ground Vehicle Technology V.* Bd. 5083, 2003, S. 52–62

[37] YE, C. ; BORENSTEIN, J.: Obstacle avoidance for the segway robotic mobility platform. In: *Proc. of the American Nuclear Society Int. Conf. on Robotics and Remote Systems for Hazardous Environments*. Gainesville, USA, 2004, S. 107–114

[38] YE, C. ; BORENSTEIN, J.: T-transformation: a New Traversability Analysis Method for Terrain Navigation. In: *Proc. of the SPIE Defense and Security Symposium*. Orlando, USA, 2004

Abbildungsverzeichnis