Generierung von Roboterinstruktionen anhand erkannter Umgebungsmerkmale in einer Karte

Diplomarbeit am Fachbereich Informatik,
Universität Bremen

vorgelegt von Marco Spindler

2. Mai 2006

Gutachter: Prof. Christian Freksa
Dr. Thomas Röfer

Ich erkläre hiermit, dass ich diese Arbeit selbständig durchgeführt und keine außer den angegebenen Hilfsmitteln und Quellen verwendet habe.

Datum: _____ Unterschrift: _____

Zusammenfassung

Diese Arbeit stellt ein Verfahren vor, dass es erlaubt einen Roboter von einer Startposition zu einem Raum innerhalb einer Büroumgebung zu leiten. Bei diesem Verfahren werden Instruktionen erzeugt, die einer Wegbeschreibung ähneln, die auch von Menschen verwendet werden. Da sich Wegbeschreibungen von Menschen meistens auch auf Umgebungsmerkmale (Landmarken) beziehen, werden Landmarken benötigt. Die Identifikation von Landmarken ist somit ein Teil dieser Arbeit. In dieser Arbeit gibt es vier Typen von Landmarken. Diese Typen werden in der weiteren Arbeit Modelltypen genannt und die erkannten Landmarken Modelle. Bezüglich der erkannten Landmarken werden in dieser Arbeit Roboterinstruktionen erzeugt. Um die Instruktionen näher an die natürliche Wegbeschreibung von Menschen zu bringen, wird in Abschnitt 5.2 eine Sprache eingeführt, die Bezug auf die Landmarken nimmt. Beispielsweise lautet eine Anweisung in dieser Sprache „`Fahre zur 3. Einfahrt links.`“.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Ziel dieser Arbeit ist es, ein Verfahren zu entwickeln, dass einen Roboter durch ein Gebäude führt. Dem Roboter wird der Weg anhand von Umgebungsmerkmalen beschrieben, die aus einer Karte der Umgebung automatisch generiert werden. Anhand der extrahierten Umgebungsmerkmale wird dann eine Wegbeschreibung erstellt, die den Roboter von seiner Startposition schrittweise zu einem Raum führt. Dies stellt eine Anlehnung an Routeninstruktionen für Menschen anhand von Landmarken dar.

1.1 Motivation

Bei der Beschreibung eines Weges zu einem Ziel geht der Mensch nur selten so vor, dass er genaue Entfernungen und Drehungen angibt. Wird jemand nach dem Weg gefragt, lautet die Antwort selten: „Gehen Sie 300m geradeaus, drehen sich dann um 90° und gehen wieder 50m ...“ . Dies wäre auch nicht zweckmäßig, da die genauen Entfernungs- und Winkelangaben sich kaum umsetzen ließen. Eine Wegbeschreibung ist zweckmäßig, wenn sie auf die Umgebung Bezug nimmt, zum Beispiel „Gehen Sie geradeaus bis zum Pförtner, dann den Flur nach links.

Betrachtet man Roboter, ist die Wegbeschreibung mit den Meterangaben und Graden für diese leicht umsetzbar. Wie genau die Durchführung anschließend ist, ist von der Genauigkeit des Roboters abhängig. Wenn ein Roboter statt genannter 5m nur 4.8m fährt und sich dann dreht, kann sich vor ihm ein Hindernis befinden, das bei 5m nicht vorhanden ist. Möchte man aber, dass jeder Mensch einem Roboter eine Wegbeschreibung geben kann, führt dieser Ansatz zu Problemen. Zu den oben genannten Angaben in Meter und Grad ist der Mensch meistens nicht fähig. Dafür versteht der Roboter eine Wegbeschreibung wie sie für menschliche Kommunikation natürlich erscheint nicht.

Um diese Anweisungen umsetzen zu können, müsste der Roboter diese Anweisungen interpretieren können. Ist der Roboter dazu in der Lage, tritt das nächste Problem auf. Der Roboter muss in seinen Sensordaten die genannten Objekte (Kreuzung, Tankstelle, Straße) erkennen. Dies ist ein sehr schwieriges, ungelöstes Problem. Im Rahmen von

Routenbeschreibungen in einer Büroumgebung nehme ich mich dem an.

1.2 Zielsetzung / Szenario

In dieser Arbeit geht es um die Entwicklung eines Verfahrens, um einen Roboter von einer Startposition zu einem Raum zu führen. Hierbei sollen die Roboterinstruktionen Bezug auf die Umgebung nehmen und als Bindeglied zu natürlicher Sprache dienen können. Bei der Umgebung handelt es sich um eine Büroetage ohne Rampen oder Treppen. Um die Instruktionen mit Bezug auf die Umgebung geben zu können, werden Referenten in der Umgebung, auch Landmarken genannt, benötigt. Ein erster Schritt dieser Arbeit wird es sein, diese Landmarken zu bestimmen. Die Erkennung von Landmarken der Umgebung basiert auf einer gegebenen Karte. In dieser Arbeit wird eine schematische Karte verwendet, die Raumkarten in Gebäuden entspricht. Die Karte braucht nicht maßstabsgenau sein. Rücksicht muss bei der Karte nur auf die Breite des Roboters genommen werden. Das heißt, dass Einfahrten und Korridore eine Mindestbreite haben müssen.

Die Karte, auf der die Erkennung basiert, besteht aus einer Liste von gerichteten Linien. Durch die Richtung wird für eine Linie eine Vorderseite definiert. Die Linien von Außenwänden laufen gegen den Uhrzeigersinn. Die Linien von Innenwänden laufen mit dem Uhrzeigersinn. Das heißt, dass eine von vorne betrachtete Wand von rechts nach links läuft. Abbildung 1.1 zeigt ein Beispiel für diese Regel.

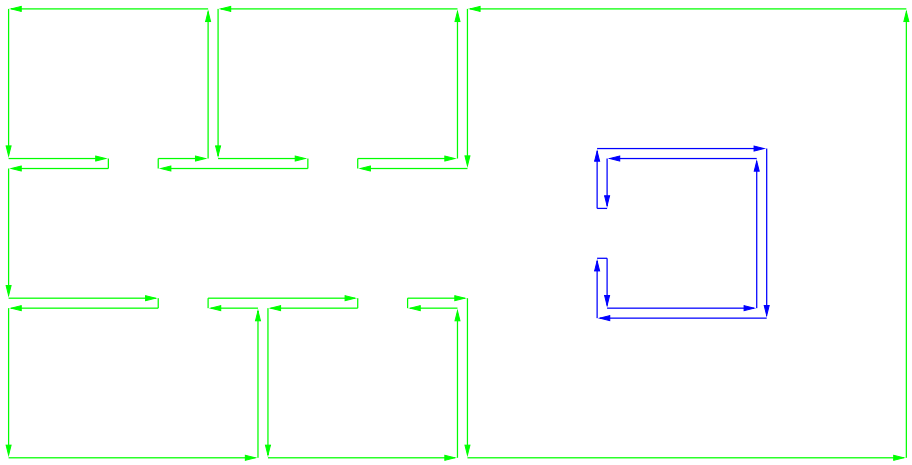

Abbildung 1.1: Karte mit Außenwänden (grün) und Innenwänden (blau), die Pfeile geben die Richtung an, in der die Wände laufen. Abbildung 4.1 in Kapitel 4 zeigt einen hierbei erkannten Korridor

In solchen Karten sollen Landmarken, etwa Korridore, erkannt werden. Anhand dieser Landmarken sollen Roboterinstruktionen erzeugt werden, die Bezug auf die Landmarken nehmen. Um den Wegbeschreibungen für Menschen gerecht zu werden, wird eine entsprechende Sprache benötigt.

1.3 Übersicht über die Arbeit

Im folgenden Abschnitt wird die Struktur der Arbeit beschrieben. Dazu wird ein kurzer Überblick geliefert, worum es in den einzelnen Kapiteln geht. Kapitel 2 liefert eine Übersicht über Arbeiten, die für meine Arbeit relevant sind. Die Arbeit besteht grob aus drei Teilen. Als erstes Landmarken, die im Anschluss an das 2. Kapitel der Einfachheit halber Modelle genannt werden. Der zweite Aspekt meiner Arbeit ist Roboterinstruktionen, und der dritte Teil ist die Generierung der Instruktionen.

Um dem Roboter eine Wegbeschreibung zu liefern, werden als erstes Umgebungsmerkmale aus einer Karte extrahiert. Jedes Umgebungsmerkmal wird einem Modelltyp zugeordnet. Ein Modell ist eine einfache Beschreibung von Umgebungsmerkmalen. Die verwendeten Modelle, wie zum Beispiel das Modell *Wand* werden in Kapitel 3 beschrieben.

Im 4. Kapitel wird die Erkennung von Modellen beschrieben. Zuerst wird dort auf die generelle Erkennung eingegangen und anschließend wird beschrieben wie die dafür benötigten Suchfunktionen erzeugt werden. Als letztes werden in Abschnitt 4.2 die erzeugten Suchfunktionen in ihrer Funktionsweise beschrieben.

Kapitel 5 beschreibt die Generierung von Roboterinstruktionen. Hierbei wird zuerst ein Graph erzeugt, der für die Pfadplanung genutzt wird. Hierfür wird für jedes Modell ein Teilgraph erzeugt, die anschließend zu einem Gesamtgraphen zusammengesetzt werden.

In Kapitel 6 wird die Evaluation des beschriebenen Verfahrens gezeigt. Als erstes werden Teile der Arbeit in simulierten Testumgebungen getestet. Zuletzt wird ein Test in der 5. Ebene des MZH an der Uni Bremen durchgeführt. Hierbei wird getestet, ob auf den Daten des Roboters einzelne Modelle erkannt werden können.

Das 7. Kapitel liefert eine Zusammenfassung und Bewertung der Arbeit.

Kapitel 2

Roboterinstruktionen

In meiner Arbeit geht es um die Generierung von Roboterinstruktionen. Um die Generierung durchführen zu können, werden verschieden Elemente benötigt. Als erstes werden Landmarken zur Bestimmung der nächsten Instruktion benutzt. Außerdem sollen die Anweisungen der natürlichen Sprache näher kommen. Da in dieser Arbeit aber nicht die Aufgabe ist, natürliche Sprache zu verstehen, wird eine Sprache definiert, die menschlichen Wegbeschreibungen nahe kommt. Zuletzt sollen die Roboterinstruktionen automatisch entlang eines Pfades generiert werden. Entsprechend dieser benötigten Teile werden hier relevante Arbeiten vorgestellt.

2.1 Landmarken

In diesem Abschnitt werden relevante Arbeiten zu Landmarken vorgestellt. Landmarken bilden in dieser Arbeit die Grundlage für die Generierung von Roboterinstruktionen. Anhand der gewählten Landmarken, welche im Abschnitt 3 als Modelle eingeführt werden, wird entschieden, welche Instruktion der Roboter ausführen soll.

Michon und Denis [4] befassen sich in ihrer Arbeit mit der Frage: Wann und warum werden visuelle Landmarken bei Wegbeschreibungen benutzt? Sie berichten über zwei Studien, welche die kognitive Funktion von Landmarken untersuchen. Hierzu stellen sie fest, dass Wegbeschreibungen Aktionen vorgeben, die nur in einer bestimmten Reihenfolge zum Erfolg führen. Aktionen sind Streckenangaben und Richtungsänderungen. Richtungsänderungen setzten sich aus der Richtung, in die fortzufahren ist, und der Position, an der die Änderung stattfinden soll, zusammen.

In meiner Arbeit setzen sich Wegbeschreibungen auch aus Bewegung und Richtungsänderungen zusammen. Die Position der Richtungsänderung wird durch die erkannten Landmarken beschrieben.

Drei Elemente sollte eine Wegbeschreibung laut Michon und Denis beinhalten. Hierzu zählen Orte, auf denen sich fortbewegt wird, wie zum Beispiel Alleen, Wege oder Straßen.

Als nächstes sollte eine Wegbeschreibung metrische Reorientierungspunkte, wie „am Ende der Straße“ oder „bei Nummer 28“ enthalten. Als letztes gehören zu einer Wegbeschreibung Landmarken, die sich entlang des beschriebenen Weges befinden.

Auch in meiner Arbeit finden sich die drei Elemente wieder. Straßen oder ähnliches sind bei mir aufgrund der gegebenen Umgebung zum Beispiel Korridore. Metrische Angaben, wie „nach 200 Metern“ gibt es bei mir nicht. Dafür gibt es Angaben wie „Ende des Korridors“ oder „3. Einfahrt links“. Landmarken im Sinne von Gebäuden oder ähnlichen gibt es in einer Büroumgebung nicht, daher beschränken sich die Landmarken in meiner Arbeit zum Beispiel auf Wände oder Einfahrten.

Die von Michon und Denis durchgeführten Studien zeigen, wie wichtig Landmarken für Menschen bei der Wegbeschreibung sind. Sie helfen dabei, sich in einer unbekannten Umgebung zurechtzufinden. In der ersten Studie ging es darum, dass Personen eine Strecke lernten und anschließend Wegbeschreibungen für diese Strecke lieferten. Hierbei stellte sich heraus, dass Landmarken nicht nur an Reorientierungspunkten benutzt wurden, sondern auch an Punkten, an denen es mehrere mögliche Richtungen zur Auswahl gab. Bei der zweiten Studie wurden Probanden Wegbeschreibungen gegeben, bei denen komplett auf Landmarken verzichtet wurde. Hierbei stellte sich raus, dass die Probanden mit solch einer Wegbeschreibung große Schwierigkeiten haben.

Anhand dieser Studien lässt sich erkennen, wie wichtig Landmarken für den Menschen sind, um sich in einer Umgebung zurechtfinden zu können. Um nun einen Roboter sicher durch eine Büroumgebung zu leiten sind auch hier Landmarken nötig.

Kuipers und Byun [3] nutzen für ihren Ansatz für das Explorieren einer unbekannten Umgebung und das Lernen einer Karte ebenfalls Landmarken. Diese Landmarken sind Teil ihres Schichtenmodells, der Spatial Symantic Hierarchy.

Raubal und Winter [5] zeigen in ihrer Arbeit, wie Wegbeschreibungen, zum Beispiel von Navigationssystemen, durch Landmarken aufgewertet werden können. Um ein automatisches Einfügen der Landmarken möglich zu machen, führen sie Bewertungen für die Landmarken ein. Es werden drei Arten von Bewertungen unterschieden. Als erstes wird eine Landmarke nach dem Aussehen beurteilt. Hierzu zählen der Unterschied zum Rest der Umgebung, Form, Farbe und die Sichtbarkeit. Als zweites Kriterium wird die Bedeutung der Landmarke genutzt. Dazu zählen kulturelle sowie geschichtliche Wichtigkeit. Außerdem zählt dazu, ob das Gebäude an der Frontseite ein Schild trägt, das auf die Bedeutung hinweist. Zuletzt wird die strukturelle Attraktivität bewertet. Darunter fallen Knoten, die zum Beispiel für Autofahrer Straßenkreuzungen, für Fußgänger Plätze oder für Geschäftsreisende Flughäfen sein können.

In meiner Arbeit kommen Bewertungen für die Landmarken auch zur Geltung. Hierbei wird die Bewertung aber genutzt, um diese Landmarken zu erkennen. Da in meiner Arbeit die Wahrnehmung der Roboter eingeschränkt ist, beschränken sich die Bewertungen auf die Grundrisse der Merkmale. Hierzu zählt zum Beispiel die Breite eines Korridors. Die genaue Bewertung der verwendeten Landmarken wird im Kapitel 3 bei dem jeweiligen Modell beschrieben.

Bei der Erkennung von Positionen hilft auch die Orientierung von Landmarken [1]. Freksa beschreibt in seiner Arbeit, wo ein Punkt b in Abhängigkeit von einer Position a liegen kann. Hierbei wird vor/hinter mit links/rechts kombiniert. So liegt der Punkt b zum Beispiel „rechts von“ oder „links vor“ a. Er geht noch weiter und nimmt die Positionierung eines Punktes in Abhängigkeit einer Strecke vor. So kann ein Punkt c zum Beispiel „rechts zwischen“ a und b liegen [2].

In meiner Arbeit spielen Richtungen dahingehend eine Rolle, zwischen rechts und links unterschieden wird. Rechts und links eines Korridors liegen Einfahrten, die der Roboter durchfahren kann.

2.2 Instruktionssprache

In diesem Abschnitt werden relevante Arbeiten zu Instruktionen vorgestellt. Instruktionen stellen die Sprache dar, die für die Kommunikation mit dem Roboter benutzt wird. Die in meiner Arbeit verwendete Instruktionssprache wird im Abschnitt 5.2 beschrieben.

Das Arbeiten mit Worten ist durch die menschliche Fähigkeit, ein weites Spektrum an physischen und mentalen Aufgaben ohne irgendwelche Messungen oder Berechnungen zu lösen [8]. Einige Beispiele hierfür sind das Fahrradfahren, das Parken eines Autos oder Sprachverständnis.

In meiner Arbeit soll der Roboter nicht die menschliche Sprache verstehen lernen. Wörter spielen die Rolle von Bezeichnungen für wahrgenommene Umgebungsmerkmale und können sollten daher helfen, Aufgaben zu lösen. In meiner Arbeit wird die natürliche Sprache genutzt, um die Instruierung des Roboters für den Menschen leichter zu machen.

Raubal und Winter [5] beschreiben Instruktionen für die Erweiterung einer einfachen Wegbeschreibung zum Beispiel von einem Navigationssystem. Eine erweiterte Beschreibung besteht aus Instruktionen folgender Form:

```
[AT landmark_i] +
[TURN LEFT | RIGHT | MOVE STRAIGHT] +
{ONTO street name} +
{(PASSING | CROSSING) landmark_j}_{0...n} +
[UNTIL landmark_k].
```

Wobei „[]“ benötigte Elemente, „{}“ optionale Element, Großbuchstaben Sprachelemente und Kleinbuchstaben Variablen darstellen. Mit Hilfe dieser Sprache lassen sich Anweisungen generieren, die zum Beispiel wie folgt aussehen:

```
AT previous landmark
TURN LEFT ONTO "Stephansplatz"
UNTIL "Haas building, a dark building of architectural
significance containing a (signed) Zara shop at the right"
```

Meine in Abschnitt 5.2 beschriebene Wegbeschreibungssprache hat eine ähnliche Struktur wie die in der hier vorgestellten Arbeit. Es gibt Landmarken, zu denen der Roboter fahren soll und es gibt Bewegungsmuster. Die Einfachheit solcher Syntax erleichtert die Implementierung des „Verstehens“ auf Seiten des Roboters.

Kapitel 3

Modelle

In diesem Kapitel werden die Umgebungsmerkmale, die in meinem Ansatz erkannt werden sollen, beschrieben. Erkannte Umgebungsmerkmale werden in verschiedene Klassen unterschieden. Diese Klassen werden im Folgenden Modelle genannt. Ein Modell ist zum Beispiel eine Wand. Außerdem werden in Abschnitt 3.6 Funktionen zum Erzeugen von Modelltypen und graduellen Relationen, die im Folgenden einfach Relationen genannt werden, beschrieben. Die hier verwendeten graduellen Relationen liefern Werte im Bereich von 0 bis 1.

Aufgrund der gegebenen Umgebung eines Bürogebäudes muss überlegt werden, welche Art von Modellen in dieser Umgebung erkannt werden können. Als erstes Modell fällt einem sofort eine Wand ein. Zwei Wände, die durch eine Lücke verbunden sind, sind eine Tür oder Einfahrt. Liegen zwei Wände sich gegenüber, kann man von einem Korridor sprechen. Treffen Korridore aufeinander, ergibt sich daraus eine Kreuzung.

Aus diesen Überlegungen ergibt sich das in Abbildung 3.1 gezeigte Bild. In dieser Abbildung werden nur die nicht atomaren Elemente gezeigt. Atomare Elemente sind Linien, die aus diesem Grund in der Darstellung weggelassen wurden. So besteht eine Wand im besten Fall aus einer Linie, kann aber auch aus mehreren Linien zusammengesetzt sein. Ein Korridor besteht aus zwei Wänden, die sich gegenüber liegen. Eine Einfahrt enthält zwei Wände, die nebeneinander liegen und Platz zwischen sich haben. Eine Kreuzung besteht aus zwei langen Korridoren, die orthogonal zueinander liegen und eine bestimmte Nähe zueinander aufweisen.Jede Instanz eines Modells erhält zudem einen Wert, der angibt, wie gut die jeweilige Instanz zu der Modellbeschreibung passt.

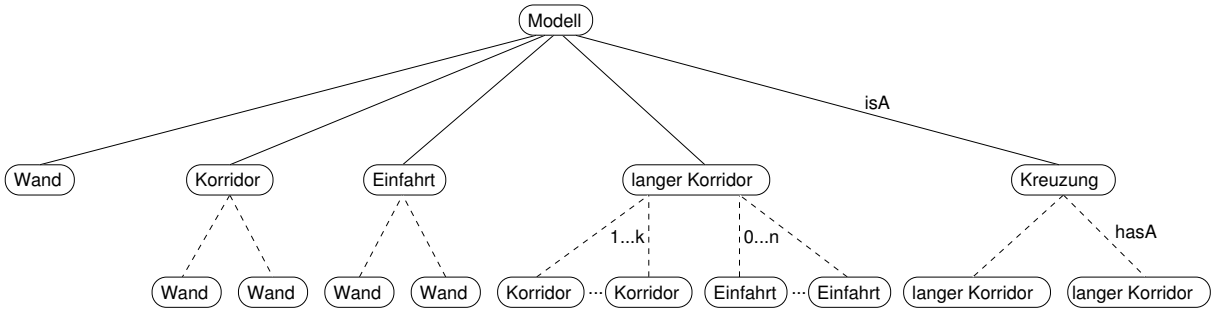

Abbildung 3.1: Das einfachste Modell ist die Wand, sie besteht nur aus einer Liste von Linien. Sowohl ein Korridor, als auch eine Einfahrt bestehen aus zwei Wänden. Ein langer Korridor besteht aus einer Menge von Einfahrten und Korridoren. Eine Kreuzung besteht aus zwei langen Korridoren.

3.1 Linien-Gruppierung

Da das einfachste Modell Wand einfach auf Linien basiert, werden die Linien einer gegebenen Karte gruppiert. Als grundlegende Gruppierungseigenschaft wird die Nähe der Linien zueinander angenommen. Nur dadurch kann gewährleistet werden, dass zusammenhängende Gruppen entstehen. Da auch in einer Gruppe wie in Abbildung 3.2 eine Wand erkannt werden soll, müssen auch Untergruppen betrachtet werden. Eine erste Lösung ist, einfach alle möglichen Untergruppen zu erzeugen. Bei einer Menge von n Linien ergeben sich 2^n Mengen (Potenzmenge). Bei Gruppen mit kleiner Anzahl von Linien ist die Rechenzeit noch akzeptabel, aber innerhalb dieser Gruppen muss weiterhin die Nähe gewährleistet sein. Somit müssen alle 2^n Mengen auf dieses Attribut hin untersucht werden. Um den potentiell riesigen Aufwand zu umgehen, wird eine zielgerichtete Gruppierung vorgenommen. Bei der Suche nach Wänden werden nur Untergruppen gesucht, bei denen die Linien nahe beieinander liegen und parallel sind. Bei dieser Parallelität werden Linien akzeptiert, die innerhalb einer bestimmten Abweichung parallel sind. Abbildung 3.3 zeigt Beispiele, bei denen **a** die Merkmale vorhanden bzw. **b** nicht vorhanden sind. Durch diese Art der Gruppierung werden im schlechtesten Fall nur n^2 Gruppen erzeugt. Dies geschieht, wenn alle Linien nebeneinander liegen und den geforderten Grad an Parallelität aufweisen. Aufgrund der großen Anzahl erzeugter Untergruppen bei der Bildung der Potenzmenge kann nicht überprüft werden, ob bei dem hier vorgestellten Verfahren Gruppen ausgelassen werden, die für die Erkennung benötigt werden. Es ist aber in keinem Experiment aufgefallen, dass eine wichtige Untergruppe ausgelassen wurde. Daher wird im Weiteren davon ausgegangen, dass keine Gruppe fehlt. Tabelle 3.1 zeigt die Zeit- und Anzahlentwicklung mit der Potenzmengenberechnung und dem hier angewandten Verfahren.

	Potenzmenge		**vorgestelltes Verfahren**	
# Linien	**Zeit**	**erz. Untergruppen**	**Zeit**	**erz. Untergruppen**
2	0 ms	$2^2 = 4$	1 ms	3
5	0 ms	$2^5 = 32$	13 ms	14
10	2 ms	$2^{10} = 1024$	48 ms	27
20	1.552 ms	$2^{20} = 1.048.576$	270 ms	86
40	???	2^{40}	1.546 ms	261

Tabelle 3.1: Zeitaufwand und Anzahl der erzeugten Untergruppen bei Bildung der Potenzmenge und dem hier vorgestelltem Verfahren.

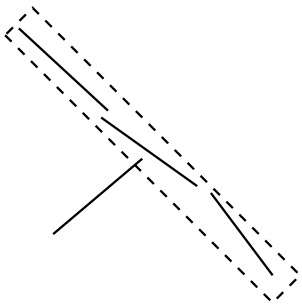

Abbildung 3.2: Mögliche Liniengruppe und die zu erkennende Wand (umrandet)

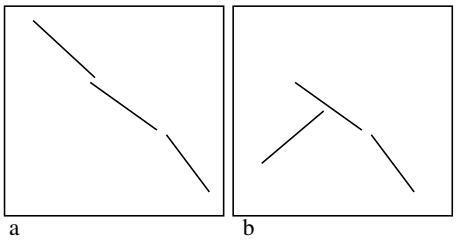

Abbildung 3.3: a. Liniengruppe, bei der die Merkmale der Nähe und der Parallelität gegeben sind. Hieraus kann ein Objekt vom Modell Wand erzeugt werden. b. Liniengruppe, bei der zwar das Merkmal der Nähe aber nicht das der Parallelität gegeben ist.

Der aus diesen Überlegungen entstandene Gruppierungsalgorithmus arbeitet nach dem folgenden Schema:

1. Aus eine Linien-Gruppe wird eine Linie $l1$ entnommen
2. $l1$ und alle Linien, die keine Parallelität zu $l1$ aufweisen, werden aus der Gruppe entfernt
3. Die restlichen Linien werden nach der Entfernung zu $l1$ sortiert
4. Nun werden aus diesen Linien Linien innerhalb eines bestimmten Radius genommen und zu $l1$ hinzugefügt. Der Radius ist in diesem Fall die Anzahl der nächsten Linien; als erstes keine Linie, damit man nur $l1$ bekommt, dann eine Linie, und so weiter. Jede zusätzlich Linie wird auf Nähe zur bisherigen Liniengruppe untersucht. Nur Linien, die eine bestimmte Nähe zur Liniengruppe haben, werden auch tatsächlich hinzugefügt.
5. Die so entstandenen Gruppen werden der Entfernung nach zu einem frei gewählten festen Punkt (hier: $x = 0, y = 0$) sortiert. Auf diese Art hat jede entstehende Gruppe eine eindeutige Reihenfolge. Entstehen zwei Gruppen mit den gleichen Linien, können doppelte Gruppen leicht über Gleichheit der Sequenz entfernt werden.
6. Wurden auf diese Art alle gültigen Nachbarn von $l1$ zur Gruppe von $l1$ hinzugefügt, wird mit der nächsten Linie aus der Ausgangsmenge fortgefahren. Die Ausgangsmenge wird dabei nicht verändert und $l1$ ist in dem neuen Durchgang in der hinzuzufügenden Gruppe enthalten.

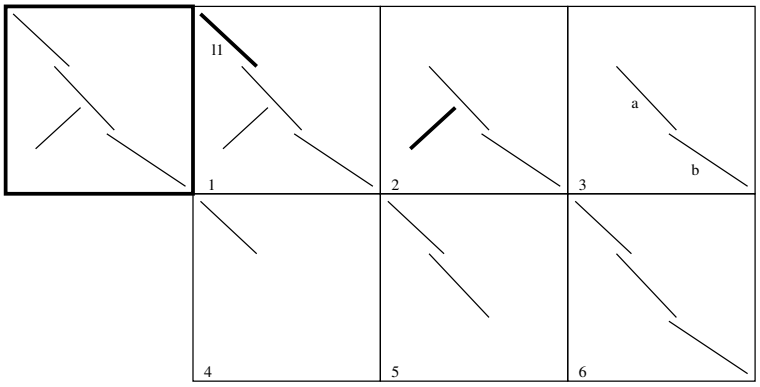

Abbildung 3.4: Fett umrandet: Ausgangsmenge; 1. Auswählen einer Linie (fett). 2. Entfernen aller Linien, die nicht Parallel sind (fett). 3. Restliche Linien nach Nähe zu Linie $l1$ sortieren. 4. Hinzufügen von 0 Linien zu $l1$ 5. Hinzufügen von 1 Linie zu $l1$ 6. Hinzufügen von 2 Linien zu $l1$. Nach jedem Hinzufügen wird die entstandene Liste von Linien gesichert. So entstehen aus der ersten Linie drei mögliche Gruppen (4., 5. und 6.).

Abbildung 3.4 zeigt beispielhaft den Ablauf des Algorithmus. Abbildung 3.5 zeigt die erzeugten Untergruppen, einer Gruppe von drei Linien. Dieses Verfahren ähnelt somit der Gruppierung gemäß der so genannten Gestaltgesetze der Wahrnehmungspsychologie [6].

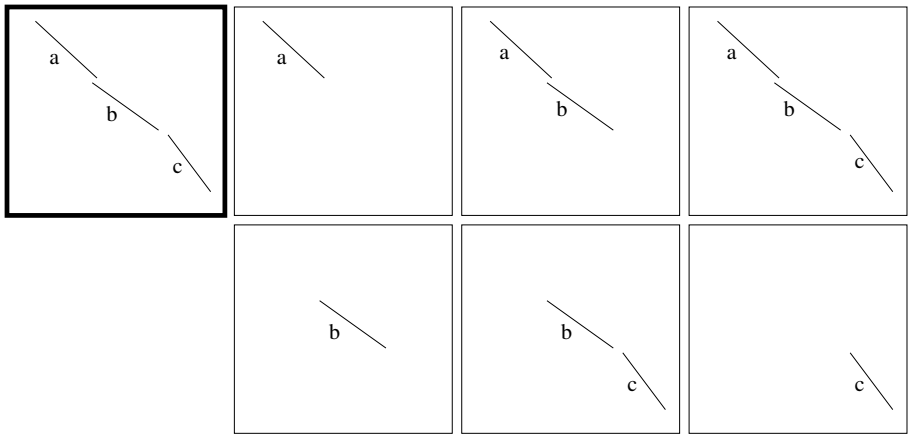

Abbildung 3.5: Links oben: Die Ausgangsmenge; in den weiteren Feldern: alle erzeugten Untergruppen.

3.2 Das Modell Wand

Eine Wand besteht aus einer Menge von Linien.

- Die Linien müssen nahe beieinander liegen.
- Die Linien müssen einen gewissen Grad an Parallelität zueinander aufweisen. Der Name dieser Relation lautet *parallel-lines*.
- Die Liniengruppe muss eine bestimmte Gesamtlänge erreichen. Der Name dieser Relation lautet *longenough*.

Bei jedem Modell gibt es für den Roboter verschiedene Möglichkeiten, damit umzugehen. Jedes Modell liefert für die Instruktionsgenerierung also einen Entscheidungspunkt, welche Instruktion ausgeführt werden soll. An einer Wand kann der Roboter nur entlang fahren.

3.2.1 Relationen des Modells Wand

Jedes Modell wird mit Hilfe von Relationen beschrieben. Jede Relation eines Modells liefert einen Wert, der zu der Entscheidung beiträgt, ob ein Kandidat zu einer Instanz eines Modells wird.

Nähe der Linien zueinander

Die Nähe der Linien zueinander wird schon durch die Gruppierung gewährleistet und braucht daher nicht weiter untersucht werden. Da die Linien der Liniengruppen nahe genug beieinander liegen wird für die Nähe keine Relation benötigt, da diese einen Wert

von 1 liefern würde. Alle nicht erzeugten Liniengruppen würden den Wert 0 erhalten und dies würde dazu führen, dass keine Wand erzeugt wird.

Parallelität der Linien

Bei dem Grad der Parallelität wird überprüft, in wie weit die einzelnen Linien untereinander parallel liegen. Um den Grad an Parallelität der gesamten Linienliste zu berechnen, wird die Parallelität jeder Linie zu jeder anderen Linie berechnet. Der Parallelitätsgrad der gesamten Liste ergibt sich aus dem Durchschnitt der einzelnen Parallelitäten. Abbildung 3.6 zeigt beispielhaft Ergebnisse dieser Relation.

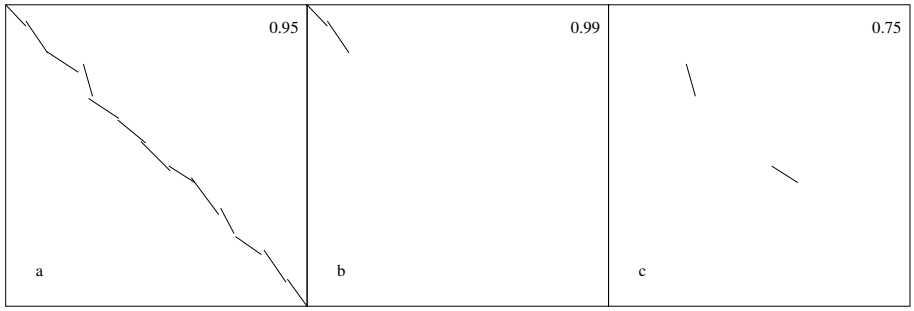

Abbildung 3.6: Parallelitäten **a**: Gesamtgrad an Parallelität aller Linien **b**: Parallelität zweier Linien, deren Grad an Parallelität größer ist als der Durchschnitt **c**: Niedrigster Grad an Parallelität der Liniengruppe

Länge der Wand

Um die Gesamtlänge der Linien in der Liste zu berechnen, wird eine Hilfslinie erzeugt. Diese Hilfslinie verbindet die beiden entferntesten Punkte miteinander. Die Länge dieser Linie bestimmt die Länge der Wand. Abbildung 3.7 zeigt eine Linienliste mit ihrer Hilfslinie und der daraus resultierenden Länge. Die Hilfslinie, die so erzeugt wurde, wird im Modell gespeichert, da sie bei Relationen der Modelle Einfahrt und Korridor auch benötigt wird, so braucht sie später nicht noch einmal berechnet werden.

3.2.2 Umgang mit dem Modell Wand

Die einzige Aktion in Verbindung mit einer Wand ist das Entlangfahren. Hierbei wird dem Roboter nur mitgeteilt, auf welcher Seite die Wand sich befindet, an der er entlang fahren soll. Eine Entfernung, die der Roboter fahren soll, wird nicht übergeben, da eine Wand nur eine gewisse Länge besitzt. Die Ausführung des Entlangfahrens endet also mit dem Ende der Wand.

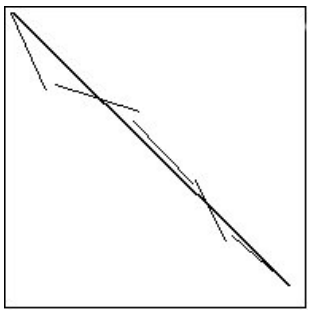

Abbildung 3.7: Länge einer Liniengruppe, gemessen anhand der Hilfslinie (dicke Linie)

3.3 Das Modell Einfahrt

Damit zwei Wände eine Einfahrt bilden können, müssen die Wände

- eine ähnliche Ausrichtung haben,
- weit genug auseinander liegen,
- zwischen sich kein weiteres Objekt haben und
- von einem bestimmten Punkt aus beide sichtbar sein.

Der Roboter soll entweder an einer Einfahrt vorbei fahren oder aber durch sie hindurch fahren.

3.3.1 Relationen des Modells Einfahrt

Ausrichtung der Wände

Für die Berechnung der Ausrichtung zweier Wände zueinander wird eine Funktion implementiert, sie heißt *beside*. Um die Ausrichtung zueinander zu berechnen, werden für beide Wände die Hilfslinien genommen. Anhand dieser Hilfslinien werden dann die Ausrichtungen überprüft. Aufgrund des gerichteten Aufbaus der Karte sollen keine Wände Einfahrten bilden, die gegenläufig sind. Abbildung 3.8 zeigt Einfahrten, die mit Hilfe dieses Kriteriums zugelassen bzw. verhindert werden. Bei Wänden mit gegenläufiger Ausrichtung würde die eine Wand von Vorne und die andere von Hinten gesehen werden. Dies wird hiermit ausgeschlossen.

Nähe der Wände zueinander

Ob die beiden Wände weit genug von einander entfernt sind, lässt sich einfach durch die Entfernung der beiden Hilfslinien zueinander festlegen.

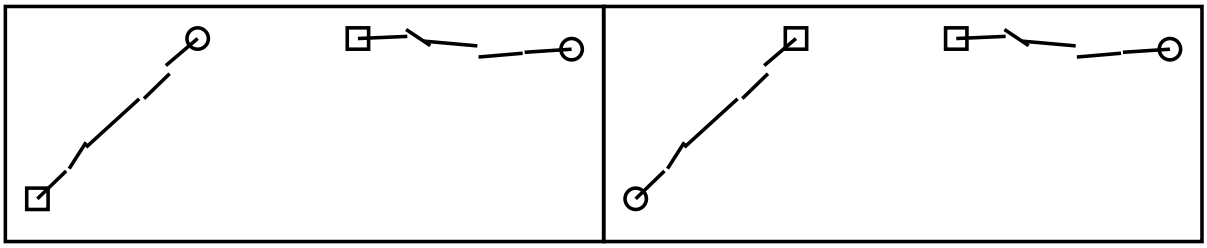

Abbildung 3.8: **links**: Erlaubte Einfahrt, **rechts**: nicht erlaubte Einfahrt (Quadrat ist Startpunkt der Wand, Kreis Endpunkt)

Erkennung von Hindernissen zwischen den Wänden

Um zu erkennen, ob sich ein Hindernis zwischen den beiden Wänden befindet, wird der Mittelpunkt zwischen den beiden Wänden berechnet. Anschließend wird überprüft, ob es eine Linie in der Karte gibt, die zu nahe an diesem Mittelpunkt liegt. Die hierfür benutzte Funktion heißt *space*.

Sichtbarkeit der Wände

Die beiden Wände, die zu einer Einfahrt werden, müssen beide gesehen werden. Hierzu wird als erstes eine Sichtposition berechnet. Dies geschieht, indem der Mittelpunkt zwischen den beiden Wänden berechnet wird. Mit Hilfe einer Linie zwischen den beiden Wänden wird eine Linksnormale berechnet. Der so erzeugte Vektor wird auf den Mittelpunkt verschoben. Der so verschobene Vektor zeigt auf den gesuchten Punkt. Von diesem Punkt aus müssen beide Wände zu sehen sein. Hierzu wird eine 180° Sichtbarkeit in Richtung der beiden Wände errechnet und überprüft, ob sich die Linien der Wände mit den so erhaltenden Polylinien schneiden. Sind die Linien Bestandteil dieser Polylinien, sollten sich die Linien komplett mit den Linien nicht nur an einem Punkt schneiden. Abbildung 3.9 zeigt eine Einfahrt mit dazugehöriger Sichtbarkeit und den Überschneidungen.

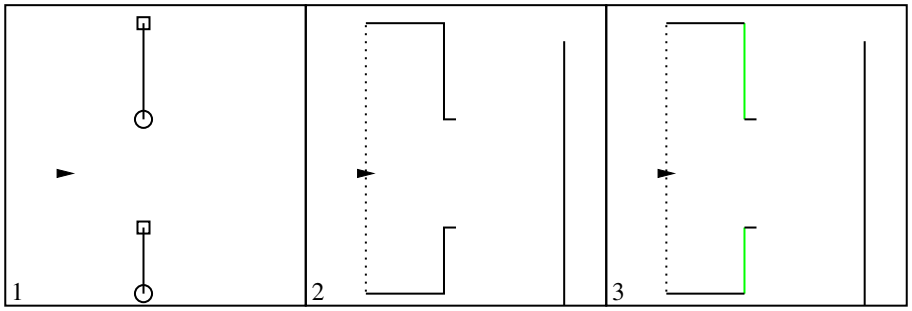

Abbildung 3.9: 1. Untersuchte Einfahrt und errechneter Sichtbarkeitspunkt (Kreis = Startpunkt, Quadrat = Endpunkt der Wand) 2. Sichtbarkeitspolylinien 3. Schnittpunkte der Linien mit den sichtbaren Linien (grün)

3.3.2 Umgang mit dem Modell Einfahrt

Die beiden Aktionen in Verbindung mit einer Einfahrt sind das Vorbeifahren und das Hineinfahren. Als erstes soll der Roboter einfach an der nächsten Einfahrt vorbeifahren, die der Roboter sieht. Es muss aber beachtet werden, dass der Roboter gerade an einer Wand entlang gefahren sein kann und somit die Einfahrt bereits neben ihm liegt. Der Roboter muss also überprüfen, ob er bereits vor einer Einfahrt steht, falls direkt neben ihm kein Hindernis ist. Außerdem ist eine Einfahrt ein Übergang; zum Beispiel in einen anderen Raum oder einen Korridor. Um dorthin zu kommen, muss der Roboter eine Einfahrt durchfahren können. Hierbei wird davon ausgegangen, dass der Roboter die zu durchfahrende Einfahrt bereits erreicht hat. Optional muss dem Roboter noch mitgeteilt werden, auf welcher Seite die zu behandelnde Einfahrt liegt, da zwei Einfahrten sich in einem Korridor gegenüber liegen können.

3.4 Das Modell Korridor

Damit zwei Wände einen Korridor bilden können, müssen diese Wände

- einen bestimmten Grad an Parallelität haben,
- sich gegenüber liegen,
- eine bestimmte Nähe zueinander haben und
- von einem zentralen Punkt aus sichtbar sein.

Der Roboter soll einen Korridor entlang fahren können.

3.4.1 Relationen des Modells Korridor

Parallelität zweier Wände

Der Grad an Parallelität der Wände zueinander wird mit Hilfe der Funktion *parallelity* berechnet, indem Hilfslinien für beide Wände genommen werden. Hier wird der Grad der Parallelität berechnet, indem die Winkeldifferenz der Ausrichtungen der Linien gebildet wird. Auf dieser Differenz wird der Kosinus angewendet. $parallelity(l1, l2) = cos(\alpha)$, wobei $l1$ und $l2$ die Hilfslinien der Wände sind. Abbildung 3.10 zeigt die zwei Hilfslinien mit dem zugehörigen Winkel α.

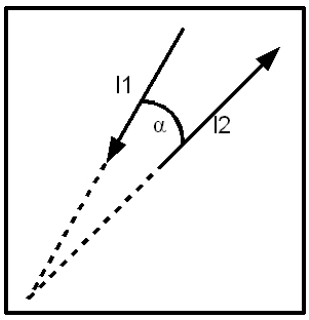

Abbildung 3.10: Parallelität zweier Wände; Ausrichtung in Pfeilrichtung

Gegenüberliegen von zwei Wänden

Um zu überprüfen, ob zwei Wände sich gegenüberliegen, werden erneut die Hilfslinien der Wände benötigt. Als erstes werden beide Linien so transformiert, dass eine Linie eine Ausrichtung von 0° und ihren Startpunkt an den Koordinaten $x = 0, y = 0$ hat. Nun wird berechnet, wo die X-Koordinaten der zweiten Linie liegen. Liegen die X-Koordinaten sowohl des Start- als auch des Endpunktes der Linie zwischen den Start- und Endpunkten der ersten Linie, liegen sich die Linien, und auch die Wände gegenüber. Diese Berechnung wird in beide Richtungen durchgeführt. Liegen die Punkte der zweiten Linie außerhalb, liegen sich die Wände nicht gegenüber. Liegt nur ein Punkt der zweiten Linie zwischen den Punkten der ersten Linie, wird berechnet, zu wie viel Prozent die Linien sich gegenüber liegen. Nachdem die Werte in beiden Richtungen berechnet wurden, wird das Maximum zurückgegeben. Abbildung 3.11 zeigt Beispiele für das Gegenüberliegen.

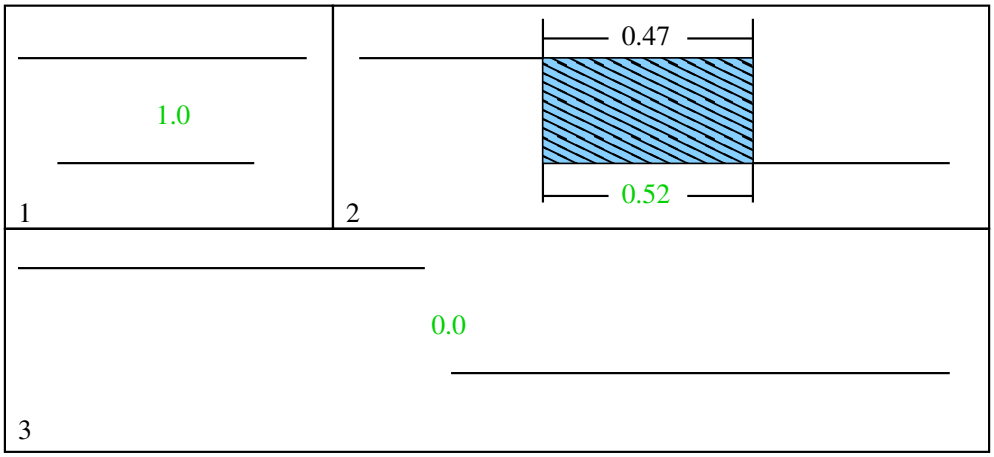

Abbildung 3.11: 1. Zwei Wände, wobei eine Wand im Bereich der anderen liegt und damit einen Wert von 1 bekommen (grün); 2. Zwei Wände, die sich zum Teil gegenüber liegen. Hierbei wird der höhere Anteil als Wert genommen (grün); 3. Zwei Wände, die sich nicht gegenüber liegen, bekommen den Wert 0 (grün).

Nähe zweier Wände zueinander

Die Nähe der Wände zueinander wird mit Hilfe der Hilfslinien berechnet. Dafür wird einfach deren Abstand berechnet.

Sichtbarkeit der beiden Wände

Für die Sichtbarkeit muss erst einmal ein Punkt berechnet werden, von dem aus beide Wände, die zu einem Korridor gehören, gesehen werden müssen. Hierzu werden zwei Linien erzeugt. Die erste Linie verbindet die Startpunkte der beiden Wände und die zweite die beiden Endpunkte. Nun wird der Schnittpunkt dieser beiden Linien berechnet. Dieser Schnittpunkt dient als Punkt für den Sichtbarkeitstest. Nun wird auf der Karte eine 360° Sichtbarkeit berechnet und auf den so erhaltenden Linien der Schnitt mit den Linien der Wände gebildet. Schneiden sich die Wände nicht nur in einem Punkt mit den Linien, sind sie von dem Sichtpunkt aus zu sehen. Abbildung 3.12 zeigt beispielhaft einen Korridor, den berechneten Sichtpunkt und sichtbare Linien mit Überschneidung.

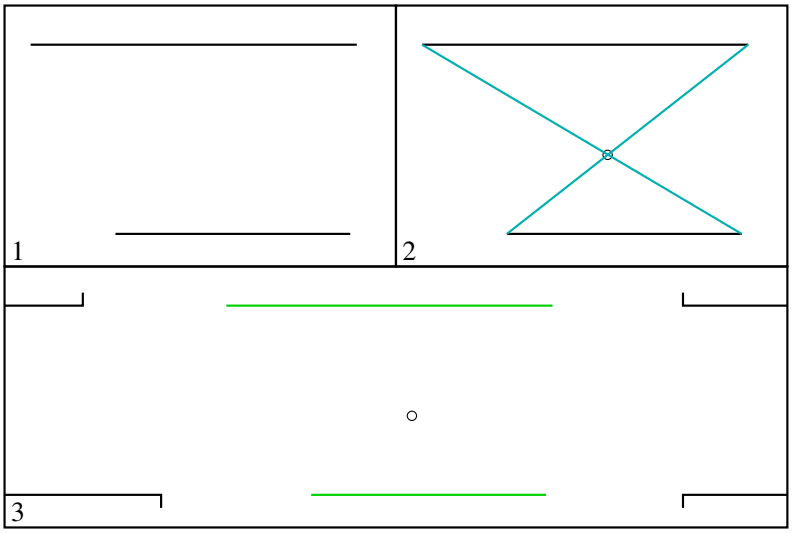

Abbildung 3.12: 1. Zu untersuchender Korridor; 2. Errechneter Sichtpunkt für den Korridor mit den erzeugten Linien (türkis); 3. Sichtpolylinien mit Überschneidungen zum Korridor (grün)

3.4.2 Umgang mit dem Modell Korridor

Der Roboter soll einen Korridor entlang fahren können. Hierbei muss beachtet werden, wie weit der Roboter fahren soll. Der Roboter soll bis zum Ende des Korridors fahren können. Er soll aber auch bis zu einer bestimmten Einfahrt fahren, um diese im Anschluss zu durchfahren. Da Korridore meist auf beiden Seiten Einfahrten haben, muss dem Roboter

noch mitgeteilt werden, auf welcher Seite sich die Einfahrt befindet, an der er anhalten soll.

3.4.3 Besonderheit des Modells Korridor

Da ein Korridor nach der Definition nur aus zwei Wänden besteht, würden viele kurze Korridore entstehen. Dies entspricht aber nicht der Vorstellung von einem Korridor in der Realität. In der Realität besteht ein Korridor aus Wänden und Türen (oder Einfahrten). Um der Realität gerecht zu werden, wird im Anschluss an das Erkennen der kurzen Korridore und dem Erkennen von Einfahrten ein Korridor erweitert. Hierzu wird ein kurzer Korridor genommen und nach Einfahrten gesucht, die eine Wand des Korridors als Teil dieser Einfahrt haben. Nun wird diese Einfahrt zum Korridor hinzugefügt und nach weiteren Einfahrten und anschließenden Korridoren gesucht, bis ein langer Korridor entsteht und keine weitere Verlängerung mehr möglich ist. Hierzu wird ein eigenes Modell, *long-corridor*, eingeführt. Dieses Modell wird benötigt, da das Modell Kreuzung auf diesem Modell aufbaut.

3.5 Das Modell Kreuzung

Eine Kreuzung besteht aus zwei langen Korridoren, die

- nahe beieinander liegen und
- in einem bestimmten Winkelbereich zueinander liegen müssen.

3.5.1 Relationen des Modells Kreuzung

Nähe zweier langer Korridore

Um zu überprüfen, ob zwei Korridore nah genug beieinander liegen, werden die Wände der beiden Korridore genommen, die am nächsten zueinander liegen. Deren Abstand wird nun als Maß der Nähe der beiden Korridore zueinander genommen. Die entsprechende Relation ist *near*.

Winkel der beiden Korridore

Die Korridore einer Kreuzung sollten in einem bestimmten Winkel zueinander liegen. Korridore, die parallel zueinander liegen, können keine Kreuzung bilden. Dies bewertet die Relation *orthogonal*. Um den Winkel zu erhalten, den die beiden Korridore zueinander haben, wird für beide Korridore eine Linie erzeugt. Diese Linie geht von einem Ende

des Korridors zum anderen Ende. Liegt die Winkeldifferenz der beiden Linien in dem geforderten Bereich, kann aus den beiden Korridoren eine Kreuzung entstehen.

3.5.2 Umgang mit dem Modell Kreuzung

Der Roboter soll an einer Kreuzung rechts oder links abbiegen können. Hierzu muss der Roboter verstehen, was rechts beziehungsweise links ist. Außerdem soll der Roboter die Kreuzung überqueren können, wenn es sich nicht um eine T-Kreuzung handelt. Hierbei ist zu beachten, wie weit der Roboter fahren soll, da kurz nach der Kreuzung bereits eine Einfahrt kommen könnte und der Roboter diese für die nächsten Schritte nicht übersehen darf.

3.6 Die Funktionen zum Erzeugen von Modelltypen und Relationen

Im Folgenden werden die Makros `defrel`, für die Erstellung neuer Relationen, und `defcat`, für die Erstellung neuer Modelltypen, beschrieben. Die Implementation wird in LISP vorgenommen. Modelle sind in dieser Implementation als Klassen definiert. Das einfachste Modell in dieser Arbeit ist die Wand. Mit Wänden könnte eine komplette Umgebung abgefahren werden. Der Roboter müsste einfach immer einer Wand folgen, dann drehen und wieder einer Wand folgen. Um einfach neue Modelltypen, wie zum Beispiel Korridore, hinzufügen zu können, gibt es das Makro `defcat`. Für neue Modelltypen werden gegebenenfalls auch neue Relationen benötigt. Auch hier soll es eine einheitliche Schnittstelle geben, um einfach neue Relationen hinzufügen zu können. Diese Aufgabe übernimmt das Makro `defrel`.

3.6.1 Das Makro `defrel`

Signatur des Makros `defrel`:

```
(defmacro defrel (name parameter &rest body))
```

Dieses Makro dient zur Erstellung und Registrierung einer neuen Relation. Es bekommt als erstes den Namen der neuen Relation übergeben. Als zweites erhält es die Parameter der Relation und zuletzt noch die Implementation der Relation. Aus diesen drei Elementen erzeugt das Makro eine Methode mit dem übergebenen Namen. Als Parameter erwartet die neue Methode die ans Makro übergebenen Parameter und führt die Berechnung anhand des ans Makro übergebenen Implementation aus. Zusätzlich zur Erstellung der neuen Methode registriert das Makro den Namen zusammen mit den Parametern in einer globalen Variable, um die neue Relation bekannt zu machen.

3.6.2 Das Makro `defcat`

Signatur des Makros `defcat`:
`(defmacro defcat (name relations objects test &rest body))`

Dieses Makro bekommt den Namen des neuen Modelltyps übergeben. Als nächstes erhält das Makro die Relationen übergeben, mit denen der neue Modelltyp beschrieben wird. Der dritte Parameter liefert die Objekte, die mit Hilfe der Relationen getestet werden sollen. Der nächste Parameter gibt einen Schwellwert an, ab dem ein neues Modell aus den übergebenen Elementen erzeugt werden soll. Der letzte Parameter dieses Makros liefert die Implementation der Relation mit deren Hilfe die Bewertung für neue Modelle vorgenommen wird.

Um zum Beispiel einen Modelltyp `wall` zu erzeugen, wird das Makro wie folgt aufgerufen.

```
(defcat wall ((longenough lines) (parallel-lines lines)) ((lines)) (> 0.85)
  (if (= 0 (longenough lines))
    0
    (parallel-lines lines)))
```

Hieraus wird eine Klasse mit dem Namen `wall` erzeugt, die von der Klasse `modell` abgeleitet ist. Diese neue Klasse bekommt die Instanzvariablen `value`, in dem die Bewertung des neuen Modells gespeichert wird, und `lines`, in dem die Liste der Linien gespeichert wird, aus denen die Wand besteht.

Außerdem wird eine neue Relation mit dem Namen `wallrel` und den `lines` als Parameter erzeugt. Diese Relation liefert die Gesamtbewertung der Linien bezüglich der Beschreibung durch die Relationen `longenough` und `parallel-lines`.

Als drittes wird eine Methode `maybe-construct-wall` erzeugt, die mit Hilfe der zuvor erstellten Relation `wallrel` und dem Schwellwert entscheidet, ob ein neues Modell erzeugt wird oder nicht. Als Parameter erwartet diese Methode `lines`. Geht der Test positiv aus, wird ein neues Objekt vom Typ `wall` zurückgegeben, sonst nichts.

Zuletzt werden der neue Modelltyp und die Konstruktionsfunktion in globalen Variablen registriert.

Kapitel 4

Erkennung von Modellen

In diesem Kapitel wird die Erkennung von Modellen in einer Karte beschrieben. Als erstes wird beschrieben, wie die Erkennung generell vonstatten geht. Anschließend wird gezeigt, wie die Erkennung umgesetzt wurde.

Hierbei wird die in Abschnitt 1.2 beschriebene Ausrichtung der Linien genutzt. Durch die gerichteten Linien wird zum Beispiel sichergestellt, dass zwei Wände, die sich gegenüberliegen und einen Korridor bilden sollen, in unterschiedliche Richtungen zeigen. Abbildung 4.1 zeigt ein Beispiel, bei dem ein Korridor erkannt werden kann und eines, bei dem kein Korridor erkannt wird.

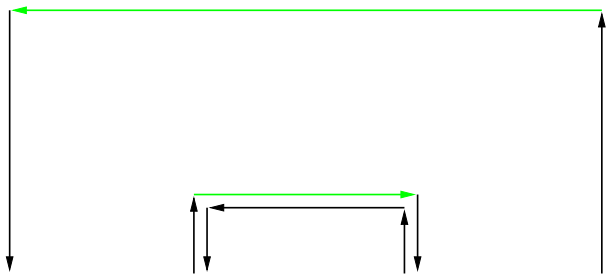

Abbildung 4.1: Ausschnitt aus der Karte in Abbildung 1.1 mit erkanntem Korridor (grün)

4.1 Erkennung

Die Erkennung von Modellen auf der Karte wird in mehreren Schritten durchgeführt. Als erstes werden Wände erkannt, da diese das einfachste Modell darstellen und nur aus einer Liste von Linien bestehen. Hierfür wird jede, durch die in Abschnitt 3.1 beschriebene Gruppierung erzeugte, Liniengruppe mit Hilfe der Relationen, die eine Wand beschreiben, überprüft. Die Werte dieser Relationen werden multipliziert, um eine Gesamtbewertung Liniengruppe zu erhalten. Der Wert, der so ermittelt wird, kann zwischen 0 und 1 liegen. Erhält eine Liniengruppe einen Wert, der größer als 0.85 ist, so wird eine Wand erzeugt.

Die Suchfunktion für das Erkennen von Wänden heißt *low-search* und wird im Abschnitt 4.2 beschrieben.

Im nächsten Schritt werden einfache Korridore und Einfahrten erkannt. Beide Modelle bestehen aus zwei Wänden. Bei dieser Suche werden die erkannten Wände genutzt. Beim Korridor werden jeweils zwei Wände mit Hilfe der, einen Korridor beschreibenden, Relationen bewertet. Hierzu werden die Ergebnisse der einzelnen Relationen miteinander multipliziert. Kommt bei der Bewertung ein Wert über 0.8 heraus, wird ein Korridor erzeugt. Bei einer Einfahrt werden zwei Wände mit Hilfe der Relationen für das Nebeneinanderliegen, Platz zwischen den Wänden, Nähe der Wände und Sichtbarkeit der Wände überprüft und bei einem Ergebnis von über 0.7 wird eine Einfahrt erzeugt. Die Suchfunktion für Korridore und Einfahrten heißt *high-search-on-wall*. Auch diese Funktion wird im Abschnitt 4.2 beschrieben.

Danach werden lange Korridore erzeugt. Hierbei werden einfache Korridore und Einfahrten in einem Modell zusammengefügt, wenn sie die gleichen Wände beinhalten. Da es sich um ein Modell handelt, das aus verschiedenen Modellen zusammengesetzt wird, gibt es keine automatisch generierte Suchfunktion. Die Funktion *make-long-corridors* wird im nächsten Abschnitt genauer beschrieben.

Als letztes werden Kreuzungen erkannt. Dies geschieht mit Hilfe der Relationen *orthogonal* und *near*. Kreuzungen bestehen aus langen Korridoren und die Suchfunktion *high-search-on-long-corridor* wird ebenfalls im nächsten Abschnitt genauer beschrieben. Mit Hilfe der Relationen wird eine Bewertung vorgenommen und, wenn diese einen Wert von 0.8 überschreitet, wird eine Kreuzung erzeugt.

Als Ergebnis der Suche wird eine Menge aller Modelle zurückgeliefert, die aus den Gruppen erzeugt werden können. Die hier verwendeten Werte zum Erzeugen von Modellinstanzen sind frei gewählt. Bei den Karten in Anhang B und C hat sich gezeigt, dass die Erkennung mit den hier gewählten Werten funktioniert. Bei der Evaluation in Abschnitt 6.2 hat sich herausgestellt, dass diese Werte gegebenenfalls angepasst werden müssen, da zum Beispiel in den Karten keine Türen verzeichnet sind, während in der realen Umgebung die Türen das Sensorbild des Roboters von dem erkannten der Karte abweichen lassen.

4.2 Funktionsweise der Suchfunktionen

Im folgenden werden die durch das Makro `mysearch` erzeugen Suchfunktionen, wie auch die Funktion zum Erstellen von langen Korridoren beschrieben. Hierbei wird als erstes auf die Funktionsweise des Makros eingegangen, bevor genauer die generierten Suchfunktionen beschrieben werden.

Im Makro `mysearch` werden als erstes aus den Konstruktoren der Modelle die Anzahl der Elemente extrahiert. Für die Suche nach Wänden gibt es nur ein Element und für die anderen Modelle gibt es zwei Elemente. Anschließend wird die Art des Modells extrahiert und dafür eine Variable angelegt. Als nächstes werden für den Funktionskörper

die Konstruktionsformen generiert. Aus diesen Teilen werden die im Folgenden genauer beschriebenen Funktionen erzeugt.

Die Funktion `lowsearch` bekommt die Liniengruppen übergeben und geht jede Gruppe einzeln durch. Auf jede Gruppe wird die `maybe-construct-`-Funktion angewendet. Diese liefert entweder ein Modell oder `nil` zurück. Ergibt sich so ein Modell, wird dieses in die vom Makro erzeugte Variable innerhalb der Funktion gespeichert und mit der nächsten Gruppe fortgefahren. Ist die Liste der Gruppen abgearbeitet, wird das Ergebnis der Suche zurückgegeben. Bei der Rückgabe handelt es sich um eine Liste, deren Element Listen sind. In jeder dieser Listen befindet sich ein Modelltyp. Im Rahmen dieser Arbeit ergibt sich daraus eine einelementige Liste, dessen Element eine Liste von Wänden ist.

Die Funktionen vom Typ *high-search-on-* bekommen eine Liste mit Elementen des Typs, der hinter dem `on` angefügt wird. In dieser Arbeit handelt es sich dabei entweder um Wände oder um lange Korridore. Diese Suchfunktionen bekommen alle Modelle ihres Typs übergeben. Aus dieser Liste werden jeweils zwei Elemente genommen und untersucht, ob ein neues Modell erzeugt werden kann. Im Fall von `high-search-on-wall` wird mit den beiden Wänden versucht erst eine Einfahrt zu erzeugen und anschließend ein einfacher Korridor. Ist das Erzeugen einer Einfahrt bereits erfolgreich, wird nicht mehr versucht ein Korridor zu erstellen, da die Anforderungen unterschiedlich sind und daher eine Einfahrt nicht auch ein Korridor sein kann. Wonach in einer solchen Funktion gesucht wird, ist abhängig von den registrierten Modelltypen. Sollte später ein weiterer Modelltyp hinzugefügt werden, der auf zwei Wänden basiert, wird nach Aufruf von *mysearch* auch nach diesem Modelltyp gesucht. Auch werden weitere Funktionen erzeugt, falls einmal ein Modelltyp eingefügt wird, der auf zwei Einfahrten oder ähnlichem basieren. Das Makro `mysearch` überprüft zuerst die eingetragenen Modelltypen und generiert anhand dieser Daten Suchfunktionen.

Kapitel 5

Generierung von Roboterinstruktionen

In diesem Kapitel wird die Vorgehensweise bei der Erzeugung von Roboterinstruktionen beschrieben. Es wird gezeigt, welche Schritte notwendig sind, damit dem Roboter eine Wegbeschreibung geliefert werden kann und wie diese aussieht. Die Syntax der Wegbeschreibung wird in der erweiterten Backus-Naur-Form angegeben.

5.1 Übersicht über die Instruktionsgenerierung

Die Erzeugung von Roboterinstruktionen findet in mehreren Schritten statt. Als erstes werden die Modelle, wie in Kapitel 4 beschrieben, für die gesamte Karte erkannt.

Die erkannten Modelle liefern die Grundlage für die Erzeugung der Roboterinstruktionen. Die Erzeugung der Instruktionen vereint die Generierung der Instruktionen mit der Pfadplanung. Die Suche basiert auf einem Graph, der mit Hilfe der Modelle erzeugt wird. Hierbei wird für jedes Modell ein Teilgraph erzeugt, der darstellt, wie der Roboter mit dem Modell umgehen kann. Zum Beispiel soll der Roboter an einer Wand entlang fahren können. Hierbei gibt es immer einen Anfang und ein Ende der Wand. Anfang und Ende einer Wand wären die beiden Knoten für einen Teilgraphen und die Tätigkeit des Entlangfahrens wird durch die Kante zwischen den beiden Knoten beschrieben. Die genaue Erzeugung der Teilgraphen wird in Abschnitt 5.3.1 beschrieben.

Wurden nun alle Teilgraphen zu den Modellen erzeugt, müssen diese miteinander verbunden werden, um den Gesamtgraphen zu erhalten. Hierbei müssen zum Beispiel die Graphen zweier Korridore über eine Kreuzung miteinander verbunden werden, wenn die Korridore auch in der Realität eine Kreuzung bilden. Hierbei ist ein erstes Kriterium die Sichtbarkeit. Des weiteren werden nur Graphen verbunden, die sich in direkter Nachbarschaft befinden.

Die Kanten des Graphen sind sowohl mit der Entfernung der Knoten zu einander als

auch mit der benötigten Tätigkeit des Roboters, um von einem Knoten zum nächsten zu kommen, annotiert. Hierdurch brauchen nur die Kanten entlang des Pfades ausgelesen werden, um die Roboterinstruktionen zu erhalten. Die Kante in Abbildung 5.1 bekommt die Bezeichnung „`Fahre links entlang der Wand.`“, was der Syntax der Wegbeschreibungssprache, beschrieben in Abschnitt 5.2, entspricht.

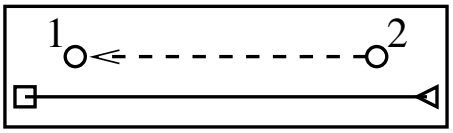

Abbildung 5.1: Eine Wand mit Anfangs- und Endpunkt an der Wand (Quadrat Wandanfang, Dreieck Wandende) Die Kante 1 ← 2 bekommt die Bezeichnung „`Fahre links entlang der Wand.`“

5.1.1 Knoten und Kanten

In dieser Arbeit stellen die Knoten einen geometrischen Punkt, ein Modell und gegebenenfalls auch Untermodelle dar. Der Punkt und das Modell werden in jedem Knoten angegeben. Das Untermodell kann zusätzlich angegeben sein, wenn das Modell Untermodelle enthält auf die sich der Knoten bezieht. Die Kanten in dieser Arbeit beinhalten eine Aktion und die Länge des korrespondierenden Weges. Im Folgenden wird genauer auf die Funktion der einzelnen Teile eingegangen.

Der Knoten

Ein Knoten ist ein Dreiertupel, der einen Punkt, ein Modell und eine Liste von Untermodellen enthält. Der Punkt eines Knoten bezieht sich auf einen korrespondierenden Punkt in der Karte. Mit Hilfe des Punktes kann die relative Position eines Knoten zu einer Kante berechnet werden. Dies ist nötig, um zum Beispiel Drehrichtungen für den Roboter zu bestimmen. Für Punkte in der Karte gibt es keine eindeutige Zuordnung von Knoten.

Das nächste Element eines Knoten ist ein Modell, dem der Knoten eindeutig zugewiesen wird. Das Modell wird gespeichert, da so einfach festgestellt werden kann, welche Knoten zu dem Teilgraphen eines Modells gehören. Das Modell wird zum Beispiel auch verwendet, um Teilgraphen zusammenzufügen.

Als letztes kann ein Knoten ein Untermodell beinhalten. Die Untermodelle helfen zum Beispiel dabei, die Teilgraphen von Korridoren und Einfahrten zu verbinden, da jeder Punkt, der sich innerhalb eines Korridorgraphen auf eine Einfahrt bezieht, die Einfahrt als Untermodell erhält. Genau beschrieben werden diese Zusammenhänge im Abschnitt 5.3.1.

Die Kante

Die Kante wird mit einer Aktion als Bezeichnung annotiert. Sie dient dazu auszugeben, was gemacht werden muss, um von einem Knoten zum nächsten zu gelangen. Die verschiedenen Arten der Bezeichner wird für die einzelnen Teilgraphen beschrieben (s. Abschnitt 5.3.1). Als zweiten Wert erhalten die Kanten die Länge des korrespondierenden Weges.

5.2 Syntax der Wegbeschreibungssprache

Nachstehend sieht man die erweiterte Backus-Naur-Form der Wegbeschreibungssprache. Siehe auch Anhang A. Jede Wegbeschreibung besteht aus mehreren Schritten:

```
Wegplan ::= {Schritt}.
```

Es kann auch sein, dass es für ein Ziel keine Wegbeschreibung gibt, daher soll es zulässig sein, dass eine Wegbeschreibung auch aus keinem Schritt besteht. Ein Schritt selber kann sich auf eine Wand, eine Einfahrt, einen Korridor oder eine Kreuzung beziehen. Außerdem soll dem Roboter mitgeteilt werden, dass er sich ausrichten soll:

```
Schritt ::= Wandschritt | Einfahrtsschritt | Drehung |
            Korridorschritt | Kreuzungsschritt.
```

An einer Wand kann der Roboter entlang fahren. Hierbei muss dem Roboter mitgeteilt werden, auf welcher Seite von ihm die Wand liegt:

```
Wandschritt ::= "Fahre " Seite " entlang der Wand.".
Seite       ::= "rechts" | "links".
```

An einer Einfahrt kann der Roboter entweder vorbei- oder in sie hinein fahren.

```
Einfahrtsschritt ::= Hineinfahren | Vorbeifahren.
Hineinfahren ::= "Fahre durch die Einfahrt.".
Vorbeifahren ::= "Fahre an der Einfahrt vorbei.".
```

Bei einem Korridor soll der Roboter diesem bis zu einem bestimmten Ziel folgen. Dieses Ziel kann eine Einfahrt, eine Kreuzung oder das Ende des Korridors sein. Befindet sich der Roboter in einem Korridor, soll ihm auch gesagt werden, dass er in eine bestimmte Einfahrt fahren soll. Hierfür benötigt die Sprache einen Konstrukt von Zahlen. Da dem Roboter Richtungsänderungen mitgeteilt werden, spielt die Seite, auf der die Einfahrt liegt hier keine Rolle. Bei einer Einfahrt ist auch noch wichtig, auf welcher Seite des Korridors diese Einfahrt liegt:

```
Korridorschritt ::= "Fahre bis " Ziel ".".
Ziel ::= "zur " Zifferohne0 {Ziffer} (". Kreuzung " | ". Einfahrt " Seite) |
         " zum Ende". Zifferohne0 ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
```

```
Ziffer ::= "0" | Zifferohne0.
```

An einer Kreuzung soll der Roboter entweder nach rechts oder links abbiegen, oder geradeaus weiter fahren:

```
Kreuzungsschritt ::= "Fahre an der Kreuzung " ("nach " Seite | "geradeaus") ".".
```

Bei Richtungsänderungen bei der Pfadplanung muss dem Roboter mitgeteilt werden, in welche Richtung er sich drehen soll. Hierbei soll der Roboter sich entweder in eine Richtung (ca. 90°) oder sich umdrehen:

```
Drehung ::= "Drehe dich " ("nach " Seite | "um") ".".
```

5.3 Vorbereitungen zum Planen eines Pfades

Im folgenden Abschnitt wird die Erzeugung des Graphen für die Wegplanung und die Erzeugung der Roboterinstruktionen beschrieben. Im ersten Abschnitt geht es dabei um die Erzeugung der Teilgraphen mit Hilfe der Modelle und im zweiten Abschnitt wird beschrieben, wie die Teilgraphen zu einem Graphen zusammengefügt werden.

5.3.1 Erzeugung von Teilgraphen auf den Modellen

Generell wird für jedes Modell ein Teilgraph erzeugt. Die einzelnen Graphen der Modelle werden im Folgenden beschrieben.

Teilgraph des Modells Wand

Die einzige Aktion, die der Roboter mit einer Wand durchführen soll, ist das Entlangfahren. Dieser Umstand soll sich auch in dem zu einer Wand korrespondierenden Graphen widerspiegeln. Das Ziel des Entlangfahrens ist das Ende einer Wand. Dieses Ziel ist ein erster Punkt für den Graphen. Da an einer Wand in beide Richtungen entlanggefahren werden kann, ergeben sich zwei Zielpunkte. Nun soll der Graph auch noch abbilden, dass der Roboter nicht auf der Wand fährt, sondern einen Abstand zu der Wand hält. Daher müssen die beiden Punkte noch verschoben werden. Als erstes werden die Punkte von der Wand weg verschoben. Um bei konkaven Ecken keine Probleme zu bekommen, werden die Punkte noch in Richtung Mitte der Wand verschoben. Hierdurch ergeben sich die Punkte in Abbildung 5.2. Sehr spitze Winkel können bei der Erzeugung des Gesamtgraphen ein Problem werden. Dieses Problem wird in Abschnitt 5.3.2 behandelt.

Um aber einen Graphen erzeugen zu können werden für jede Kante ein Start- und ein Zielknoten benötigt. Da nun zwei Punkte existieren, die für den Graphen die Knoten liefern können, brauchen die Punkte nur noch in Knoten umgewandelt werden. Hierzu wird ein Knoten erzeugt, der als Punkt einen der beiden Punkte erhält und dessen Modell

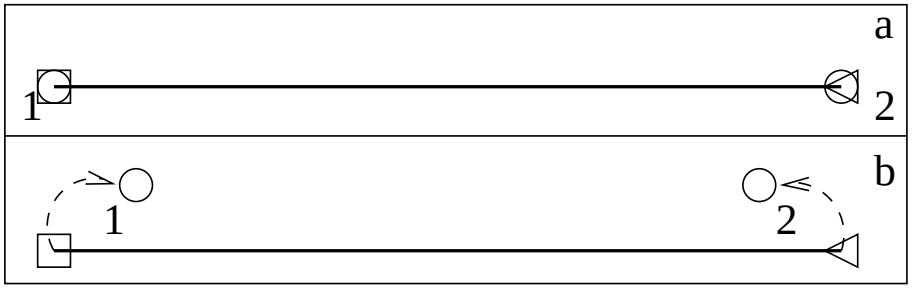

Abbildung 5.2: a) Wand mit Zielpunkten am Ende der Hilfslinien; b) Wand mit den verschobenen Zielpunkten

die Wand selber wird. Wie schon erwähnt kann in zwei Richtungen entlang der Wand gefahren werden. Dies soll sich auch in der Anzahl der Kanten zeigen. Der Unterschied dieser beiden Kanten liegt in deren Bezeichnung, welche die Aktion angibt, die der Roboter ausführt, um von einem Knoten zum Nachfolger zu gelangen. Abbildung 5.3 zeigt die erzeugten Kanten. Die Bezeichnung der Kanten wird mit Hilfe der Orientierungen der Wand und der Kante ermittelt. Da die Vorderseite links zur Richtung der Wand liegt, ist die Wand bei gleicher Orientierung der Kante auf der rechten Seite. Hierdurch ergibt sich für die Kante von Punkt 1 zu Punkt 2 die Bezeichnung „`Fahre rechts entlang der Wand.`“. Für die Kante von Punkt 2 zu Punkt 1 ergibt sich die Bezeichnung „`Fahre links entlang der Wand.`“

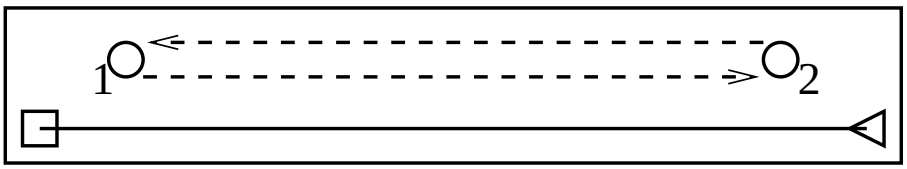

Abbildung 5.3: Wand mit Graphen,
$1 \rightarrow 2$ „`Fahre rechts entlang der Wand.`“,
$2 \rightarrow 1$ „`Fahre links entlang der Wand.`“

Teilgraph des Modells Einfahrt

Der Roboter soll an einer Einfahrt vorbei oder durch sie hindurch fahren. Dies soll auch durch einen Graphen dargestellt werden. Da die Einfahrt aus zwei Wänden besteht, zwischen denen ein gewisser Abstand liegt, ist der Roboter an der Einfahrt vorbei, sobald er eine der beiden Wände erreicht. Hierdurch ergeben sich zwei Zielpunkte. Für das Hindurchfahren wird ein Punkt benötigt, der hinter der Einfahrt liegt. Hierzu wird der Punkt bestimmt, der mittig zwischen den begrenzenden Wänden liegt. Diese drei Punkte zeigt Abbildung 5.4.

Wie auch schon bei dem Wandgraphen müssen die Punkte, welche die Enden der Einfahrt markieren, vor die Einfahrt verschoben werden. Der dritte Punkt soll hinter der

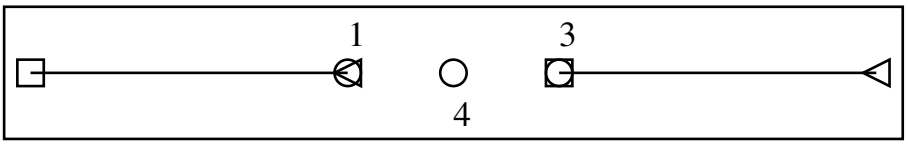

Abbildung 5.4: Eine Einfahrt mit Anfangs- und Endpunkt an der Wand (Quadrat Wandanfang, Dreieck Wandende)

Einfahrt liegen, um das Ziel für das Hindurchfahren zu sein. Nun existieren bereits drei Punkte, die als mögliches Ziel fungieren. Die Zielpunkte sind die ersten Knoten innerhalb des Graphen für die Einfahrt. Die Zielpunkte für die Begrenzung der Einfahrt könnten direkt miteinander verbunden werden, da von einem der beiden Punkte zum nächsten das Vorbeifahren als Behandlung der Einfahrt reicht. Jetzt wäre es möglich, beide Punkte auch mit dem Ziel für das Hindurchfahren zu verbinden. Davon wird Abstand genommen, da das Verbinden zweier Einfahrtsgraphen erschwert werden würde. Daher wird für das Ziel des Hindurchfahrens noch ein Startknoten eingefügt. Dieser wird direkt vor der Einfahrt zwischen den beiden Begrenzungspunkten der Einfahrt platziert. Hierdurch bekommt der Graph einer Einfahrt vier Knoten. Abbildung 5.5 zeigt die Einfahrt mit ihren vier Punkten.

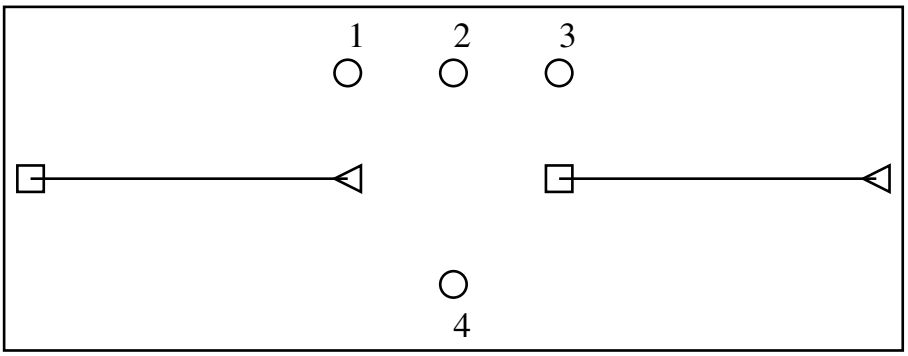

Abbildung 5.5: Einfahrt bei der die Zielpunkte bereits an der Position für den Graphen sind

Aus diesen Punkten werden jetzt die Knoten des Graphen erzeugt. Als letzter Schritt müssen noch die Kanten in diesen Graphen eingefügt werden. Hierbei werden erst einmal zwei „`Lambda`“-Kanten eingefügt: (1) → (2) und (3) → (2). Dies geschieht, da es keine Aktion „Fahre mittig vor die Einfahrt.“ gibt. Um das Vorbeifahren darzustellen, werden Kanten eingefügt, die von dem Mittelpunkt zu den Begrenzungspunkten führen. Für die Bezeichnung werden die Orientierungen der Kante und der Einfahrt zu Hilfe genommen. Bei gleicher Orientierung der Kante und der Einfahrt ergibt sich die Bezeichnung „`Fahre rechts an der Einfahrt vorbei.`“. Dies trifft auf die Kante (2) → (3) zu. Für die Kante (2) → (1) ergibt sich so die Bezeichnung „`Fahre links an der Einfahrt vorbei.`“. Als letztes wird die Kante (2) → (4) für das Hindurchfahren erzeugt. Hierbei lautet die Bezeichnung: „`Fahre durch die Einfahrt.`“. Der vollständige Graph ist in Abbildung 5.6 zu sehen.

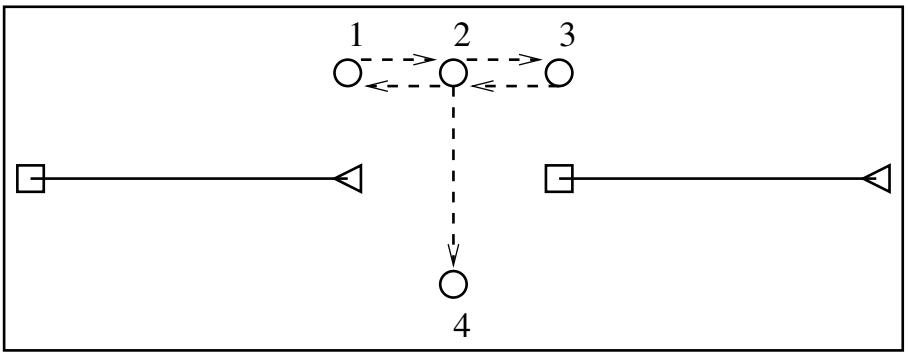

Abbildung 5.6: Einfahrt mit Graphen,
$1 \rightarrow 2$, $3 \rightarrow 2$ „`Lambda`“,
$2 \rightarrow 3$ „`Fahre rechts an der Einfahrt vorbei.`“,
$2 \rightarrow 1$ „`Fahre links an der Einfahrt vorbei.`“,
$2 \rightarrow 4$ „`Fahre durch die Einfahrt.`“

Teilgraph des Modells Langer Korridor

Da jeder kurze Korridor in einen langen Korridor überführt wird, benötigt der kurze Korridor keinen eigenen Graphen. Für das Erzeugen der Teilgraphen spielen nur die im Folgenden beschriebenen langen Korridore eine Rolle.

Bei einem Korridor kann der Roboter bis zu einer bestimmten Einfahrt oder ans Ende fahren. Dieser Umstand soll sich auch im Graphen des Korridors widerspiegeln. Zwei Punkte hierfür sind recht einfach zu finden. Als Punkte, welche die Enden des Korridors markieren, werden die Punkte von dessen Hilfslinie benutzt. Um die Position für die Einfahrten zu bekommen, werden einfach die Mittelpunkte der Einfahrten benutzt. Einen Korridor mit den so gewonnenen Punkten zeigt Abbildung 5.7.

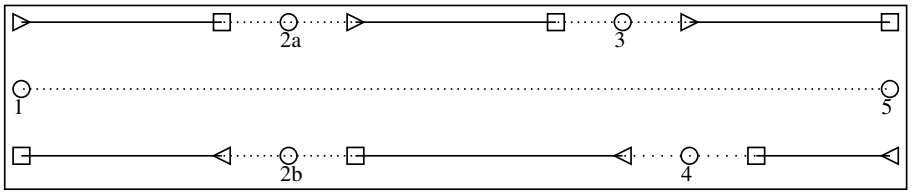

Abbildung 5.7: Ein Korridor mit bestehender Hilfslinie zwischen Punkt 1 und Punkt 5, sowie den Linien zur Mittelpunktfindung für die Punkte 2a, 2b, 3 und 4

Nun soll der Roboter aber von einer Einfahrt zur nächsten fahren. Dies soll auch im Graphen dargestellt werden. Hierfür werden die Punkte der Einfahrten Richtung Mitte des Korridors verschoben. Aus diesen Punkten werden nun Knoten erzeugt. Jeder Knoten erhält einen Punkt, den Korridor als generierendes Referenzmodell und, wenn es sich nicht um einen Endpunkt handelt, die zugehörige Einfahrt als Untermodell zugeordnet. Ergibt es sich, dass, wie in der Abbildung zu sehen, zwei Punkte an genau der gleichen Position liegen, dann werden diese Punkte zu einem Knoten zusammengefasst. Hierbei erhält der

Knoten dann aber beide Einfahrten als Untermodelle. Durch dieses Verschieben und das Erzeugen der Knoten ergeben sich die in Abbildung 5.8 gezeigten Knoten.

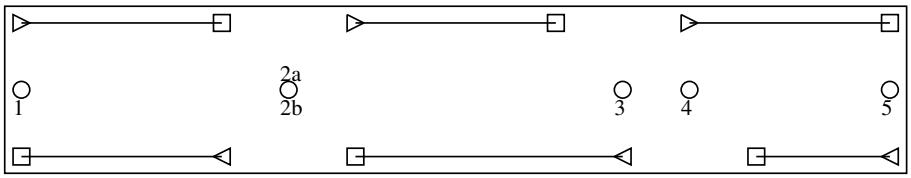

Abbildung 5.8: Korridor bei dem alle Zielpunkte bereits an der Position für den Graphen sind

Zuletzt müssen noch die Kanten mit den Bezeichnungen für die Aktionen erzeugt werden. Hierfür wird bei einem Endknoten angefangen und die anderen Punkte nach der Entfernung zu diesem Endpunkt sortiert. Dadurch können einfach der Reihenfolge nach die Kanten erzeugt werden. Um zu entscheiden, welche Bezeichnung eine Kante bekommt, wird die Ausrichtung des Untermodells vom Zielknoten benutzt. Hat ein Knoten kein Untermodell, handelt es sich um einen Endknoten. In diesem Fall lautet die Bezeichnung „`Fahre zum Ende des Korridors.`“. Hat ein Knoten mehrere Untermodelle, liegen zwei Einfahrten direkt gegenüber. Dadurch ergibt sich die Bezeichnung „`Fahre zur 1. Einfahrt rechts/links.`“. Diese Bezeichnung wird nur innerhalb des Graphen verwendet und wird nach der Berechnung des Pfades bei der Zusammenfassung der Instruktionen ersetzt. Welche Einfahrt später wichtig ist, wird durch die Pfadplanung entschieden. Hat der Zielknoten nur ein Untermodell, handelt es sich um einen einfachen Einfahrtknoten. Hier wird die Orientierung der Einfahrt mit der der erzeugten Kante verglichen. Haben beide die gleiche Orientierung, lautet die Bezeichnung „`Fahre zur 1. Einfahrt rechts.`“. Sind die Orientierungen gegenläufig, ergibt sich die Bezeichnung „`Fahre zur 1. Einfahrt links.`“. Abbildung 5.9 zeigt den Korridor mit seinem Graphen. Die Punkte 2a und 2b wurden zu Knoten 2 zusammengefasst.

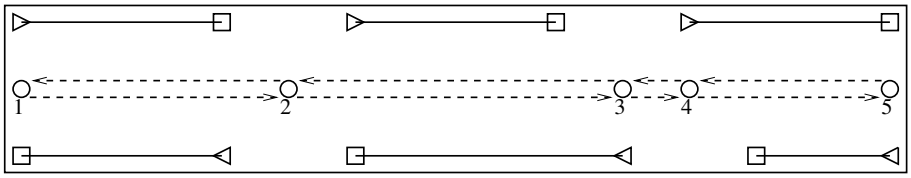

Abbildung 5.9: Korridor mit Graphen,
$1 \rightarrow 2$, $3 \rightarrow 2$ „`Fahre zur 1. Einfahrt rechts/links.`“,
$2 \rightarrow 1$, $4 \rightarrow 5$ „`Fahre zum Ende des Korridors.`“,
$2 \rightarrow 3$, $5 \rightarrow 4$ „`Fahre zur 1. Einfahrt links.`“,
$3 \rightarrow 4$, $4 \rightarrow 3$ „`Fahre zur 1. Einfahrt rechts.`“

Teilgraph des Modells Kreuzung

Kreuzungen können in zwei Grundformen auftreten. Einmal als T-Kreuzung und einmal als X-Kreuzung. Beim Erstellen eines Teilgraphen für eine Kreuzung wird auf diesen Um-

stand eingegangen. Die Kreuzungstypen unterscheiden sich durch die Nähe der Hilfslinien der Korridore zueinander. Bei einer X-Kreuzung überschneiden sich die beiden Hilfslinien. Bei einer T-Kreuzung gibt es keinen Schnittpunkt der beiden Hilfslinien. Als erstes wird die Erzeugung eines Teilgraphen für eine X-Kreuzung beschrieben. Der zweite Teil dieses Abschnittes beschäftigt sich dann mit dem Graphen einer T-Kreuzung.

X-Kreuzung Bei einer X-Kreuzung schneiden sich zwei Korridore. Da jedes Modell eine Hilfslinie besitzt, können die Hilfslinien benutzt werden, um den Schnittpunkt dieser beiden Korridore zu bestimmen. Dies geschieht, um den Mittelpunkt der Kreuzung zu bestimmen. Dieser so gewonnene Punkt liefert die Grundlage für alle Knoten dieses Graphen und ist in Abbildung 5.10 zu sehen.

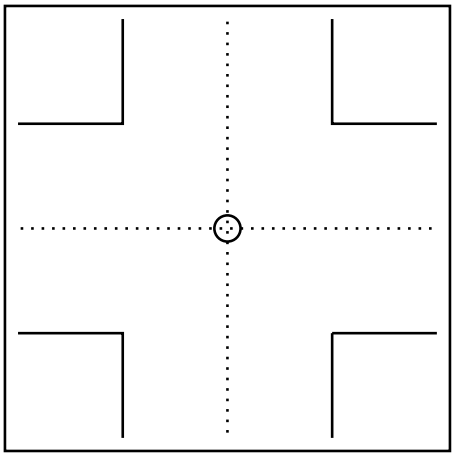

Abbildung 5.10: Kreuzung mit Mittelpunkt, welcher der Schnittpunkt der Hilfslinien der Korridore entspricht.

Mit Hilfe dieses Punktes werden vier Knoten erzeugt, die so platziert werden, dass immer zwei Knoten auf der Hilfslinie eines Korridors liegen. Die Punkte werden so platziert, dass der Mittelpunkt direkt in der Mitte zwischen den gerade platzierten Punkten liegt. Aus diesen vier Punkten werden Knoten erzeugt, die als Modell die Kreuzung und als Untermodell den jeweiligen langen Korridor bekommen. Der Mittelpunkt wird anschließend nicht mehr benötigt. Die so erzeugten Knoten sieht man in Abbildung 5.11.

Nun werden noch die Kanten erzeugt. Die Knoten, die als Untermodell den gleichen Korridor besitzen, werden mit „`Fahre geradeaus.`“ bezeichnet. Mit Hilfe der Knoten eines Korridors wird eine Hilfslinie erzeugt. Nun werden die Kanten für die Verbindung des Startknotens dieser Linie mit den beiden Knoten, die den anderen Korridor als Untermodell haben, erzeugt. Liegt der Knoten links von der Linie, wird die Kante mit „`Fahre nach links.`“ bezeichnet, im anderen Fall mit „`Fahre nach rechts.`“. Den so erzeugten Graphen zeigt Abbildung 5.12. Auch werden dort die genauen Bezeichnungen der Kanten mit angegeben.

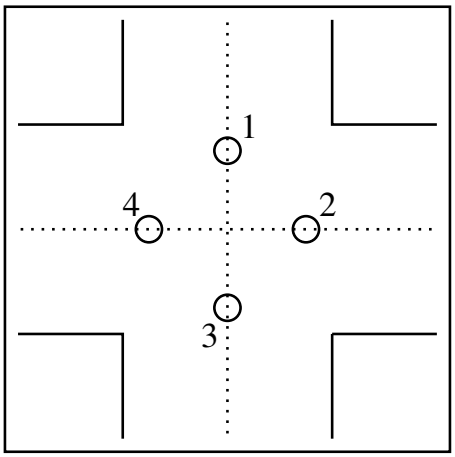

Abbildung 5.11: Kreuzung mit den Knoten auf den Hilfslinien des jeweiligen Korridors

T-Kreuzung Auch bei einer T-Kreuzung kann der Roboter nur geradeaus fahren oder abbiegen. Diese beiden Möglichkeiten werden sich auch im Graphen der Kreuzung widerspiegeln. Benötigt werden drei Punkte. Jeder Punkt gibt eine Richtung an, in der der Roboter an der Kreuzung fahren kann. Da, anders als bei der X-Kreuzung, sich die Korridore nicht kreuzen sondern nur beieinander liegen, gibt es keinen Schnittpunkt von dessen Hilfslinien. Da aber ein Endpunkt des endenden Korridors näher an dem anderen Korridor liegt als der andere Endpunkt, kann bestimmt werden, welcher Korridor der durchgehende ist. Die Hilfslinie des anderen Korridors wird in Richtung des nahen Endpunktes verlängert, um so einen Schnittpunkt mit der Hilfslinie des zweiten Korridors zu bekommen. Abbildung 5.13 zeigt diese Schritte zur Erzeugung des Schnittpunktes.

Dieser Schnittpunkt wird nun zur Erzeugung der drei, oben erwähnten, Punkte benutzt. Die ersten beiden Punkte werden vom Schnittpunkt aus auf der Hilfslinie des durchgehenden Korridors jeweils in eine Richtung verschoben. Der dritte Punkt wird in Richtung des Endpunktes des zweiten Korridors verschoben. Alle drei Punkte werden im gleichen Abstand zum Schnittpunkt, der anschließend nicht mehr benötigt wird, platziert. Aus diesen drei Punkten werden gleich Knoten für den Graphen erzeugt, die als Modell die Kreuzung erhalten. Als Untermodell erhalten die ersten beiden Punkte den durchgehenden, und der dritte Knoten den endenden Korridor. Abbildung 5.14 zeigt die drei Knoten und den Schnittpunkt.

Um den Graphen zu vervollständigen, werden zuletzt die Kanten erzeugt. Hierbei bekommen die Kanten, welche die Knoten mit dem gleichen Untermodell verbinden, die Bezeichnung „`Fahre geradeaus.`“. Die Knoten 1 und 2 in Abbildung 5.14 werden zum Beispiel auf diese Weise verbunden. Um auch Knoten 1 mit Knoten 3 zu verbinden, wird überprüft, ob Knoten 3 rechts oder links der Linie von Knoten 1 zu Knoten 2 liegt. Liegt Knoten 3 rechts von der Linie, bekommt die Kante die Bezeichnung „`Fahre nach rechts.`“. Andernfalls lautet die Bezeichnung „`Fahre nach links.`“. Aufgrund der Positionierung der Punkte kann auch gleich die Kante von Knoten 3 zu Knoten 1 erzeugt werden. Liegt Knoten 3 rechts von der Linie, dann liegt Knoten 1 links von Knoten 3 und

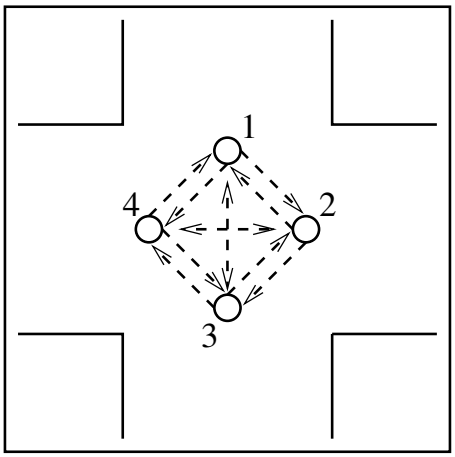

Abbildung 5.12: Kreuzung mit Graphen,
$1 \rightarrow 2$, $2 \rightarrow 3$, $3 \rightarrow 4$, $4 \rightarrow 1$ „`Fahre nach links.`“,
$2 \rightarrow 1$, $1 \rightarrow 4$, $4 \rightarrow 3$, $3 \rightarrow 2$ „`Fahre nach rechts.`“,
$1 \rightarrow 3$, $2 \rightarrow 4$, $3 \rightarrow 1$, $4 \rightarrow 2$ „`Fahre geradeaus.`“

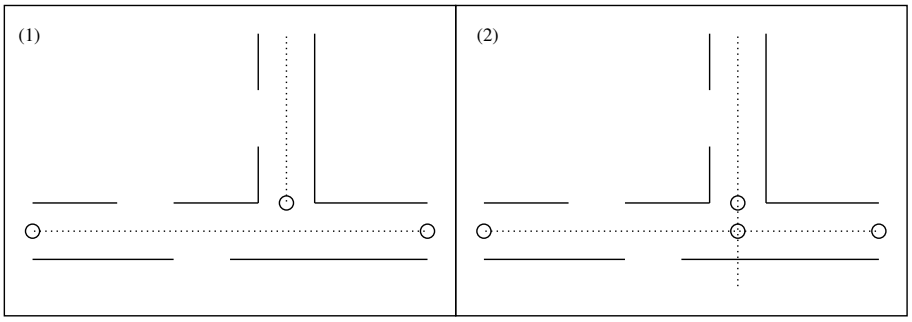

Abbildung 5.13: T-Kreuzung mit (1) den Endpunkten des durchgehenden und dem nahen Endpunkt des endenden Korridors, (2) verlängerte Hilfslinie des endenden Korridors und dadurch erzeugten Schnittpunkt

umgekehrt. Genauso wird für die Kanten zwischen Knoten 2 und Knoten 3 verfahren. Abbildung 5.15 zeigt den so entstandenen Graphen.

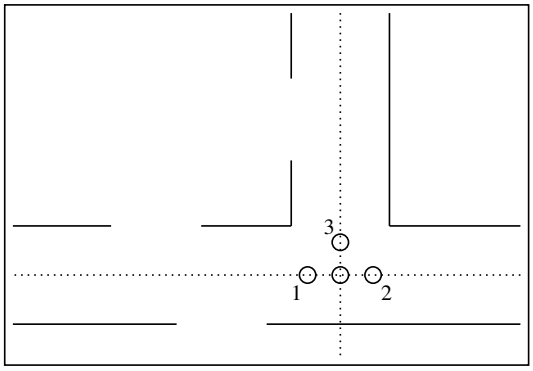

Abbildung 5.14: T-Kreuzung mit den erzeugten Knoten und dem anschließend nicht mehr benötigten Schnittpunkt

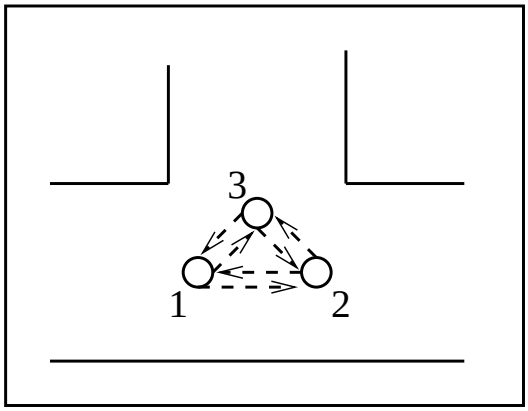

Abbildung 5.15: Kreuzung mit Graph:
$1 \rightarrow 2$, $2 \rightarrow 1$ „`Fahre geradeaus.`“,
$1 \rightarrow 3$, $3 \rightarrow 2$ „`Fahre nach links.`“,
$2 \rightarrow 3$, $3 \rightarrow 1$ „`Fahre nach rechts.`“

5.3.2 Verbinden der Teilgraphen zu einem Gesamtgraphen

Die Verbindung der Teilgraphen zu einem Gesamtgraphen läuft in mehreren Schritten ab, die im Folgenden genauer beschrieben werden. Als erstes werden Einfahrten miteinander verbunden. Als nächstes werden Einfahrten mit Korridoren verbunden, um aus einem Korridor in einen Raum oder einen anderen Korridor zu gelangen. Danach werden die Korridore einer Kreuzung über den Graphen der Kreuzung miteinander verbunden. Als letztes werden die Graphen von Wänden, die zu keinem anderen Modell gehören, in den Gesamtgraphen integriert.

Verbindung von zwei Einfahrtsgraphen

Beim Verbinden von zwei Einfahrtsgraphen werden nur die Knoten betrachtet, die als Modell eine Einfahrt besitzen. Dies reduziert die Anzahl der Knoten, die sortiert werden müssen, was den Aufwand der Suche verringert. Um zwei Einfahrten miteinander zu verbinden, müssen diese nahe beieinander liegen. Dies ist der Fall, wenn eine Einfahrt in einen Raum führt und eine zweite Einfahrt an gleicher Stelle aus dem Raum führt. Um alle Einfahrten mit ihren direkten Nachbarn zu verbinden, wird jeder Knoten einzeln betrachtet. Der betrachtete Knoten beinhaltet eine Einfahrt. Für diese Einfahrt werden alle zugehörigen Knoten herausgesucht. Diese Suche liefert vier Knoten, wobei die Knoten, von denen eine „`Lambda`“-Kante wegführt, nicht betrachtet werden brauchen, da sie nichts mit der Durchfahrt zu tun haben. Dadurch bleiben noch zwei Knoten übrig. Einmal der Zielknoten, von dem keine Kante fortführt, und der Knoten, der über eine Kante mit dem Zielknoten verbunden ist. Abbildung 5.16 zeigt diese Schritte an einem Beispiel.

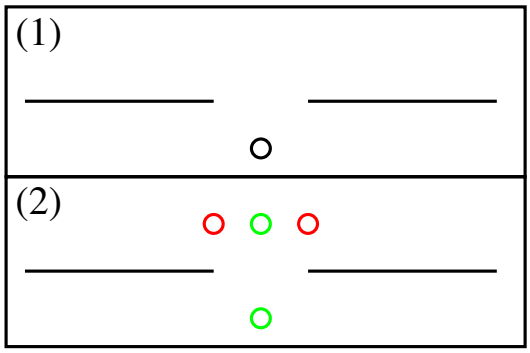

Abbildung 5.16: (1) Betrachteter Knoten mit der dazugehörigen Einfahrt; (2) Einfahrt mit allen Knoten (nicht betrachtet, betrachtet)

Der Zielknoten ist der Knoten, der für die Nähebestimmung der anderen Knoten benutzt wird. Der nächste Knoten zum Zielknoten liefert die zweite Einfahrt. Es kann aber auch sein, dass der nächste Knoten, der Knoten ist, der über eine Kante mit dem Zielknoten verbunden ist. In diesem Fall gibt es keine Nachbareinfahrt und die Knoten der betrachteten Einfahrt werden in eine Liste der geänderten Knoten eingefügt, ohne dass Änderungen vorgenommen wurden. Diese Knoten werden außerdem aus der Liste der zu beachtenden Knoten entfernt. Wird ein Knoten gefunden, der eine andere Einfahrt als Modell hat, dann werden alle Knoten für diese zweite Einfahrt gesucht. Auch hier werden die Knoten nicht beachtet, von denen eine „`Lambda`“-Kante wegführt. Übrig bleiben die selben Knoten wie bei der ersten Einfahrt. Mit diesen Knoten wird jetzt die Verbindung der beiden Teilgraphen vorgenommen. Als erstes werden die beiden Startknoten über Kanten verbunden. Beide Kanten bekommen die Bezeichnung „`Fahre durch die Einfahrt.`“. Zuletzt werden noch die beiden Zielknoten komplett entfernt, das sie durch die Startknoten der jeweils anderen Einfahrt ersetzt wurden. Abbildung 5.17 zeigt den so entstandenen Graphen, wobei nur die beiden Einfahrten betrachtet werden.

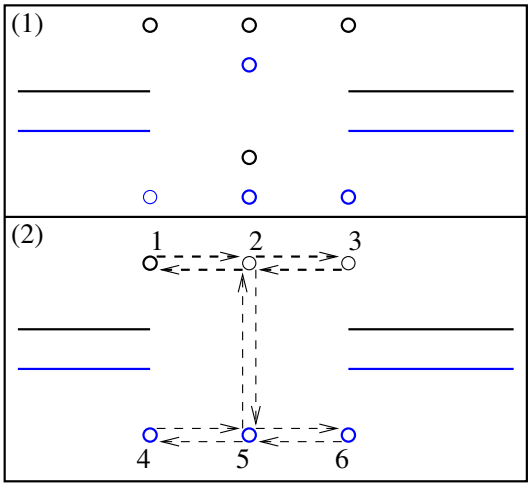

Abbildung 5.17: (1) Nahe Einfahrten mit ihren Knoten (schwarz: 1. Einfahrt, blau 2. Einfahrt);
(2) Verbundener Graph:
$1 \rightarrow 2$, $3 \rightarrow 2$, $4 \rightarrow 5$, $6 \rightarrow 5$ „`Lambda`“,
$2 \rightarrow 1$, $5 \rightarrow 6$ „`Fahre rechts an der Einfahrt vorbei.`“,
$2 \rightarrow 3$, $5 \rightarrow 4$ „`Fahre links an der Einfahrt vorbei.`“,
$2 \rightarrow 5$, $5 \rightarrow 2$ „`Fahre durch die Einfahrt.`“

Verbindung von Einfahrts- und Korridorgraphen

Ein Korridorgraph besteht aus Knoten, die den Anfang oder das Ende des Korridors markieren und aus Knoten, welche die Position von Einfahrten zeigen. Um einen Korridor mit Einfahrten zu verbinden, werden nur die Knoten benötigt, die sich auf Einfahrten beziehen. Da die benötigten Knoten als Untermodell eine Einfahrt und die Endknoten kein Untermodell haben, sind die benötigten Knoten leicht darüber zu identifizieren. Von jedem Knoten eines Korridorgraphen wird das Untermodell genommen und nach dessen Knoten gesucht. Von diesen Knoten wird nur ein Knoten benötigt. Dieser Knoten lässt sich leicht identifizieren. Von dem Knoten führt eine Kante fort, die als Bezeichnung „`Fahre durch die Einfahrt.`“ trägt. Dieser Knoten wird über eine Kante mit der Bezeichnung „`Lambda`“ mit dem betrachteten Knoten des Korridorgraphen verbunden. Die entgegengesetzte Kante bekommt die gleiche Bezeichnung. Da es vorkommen kann, dass ein Knoten zwei Untermodelle hat, wird in diesem Fall die Prozedur für jedes Untermodell durchgeführt. Abbildung 5.18 zeigt einen Korridor mit den zusätzlichen „`Lambda`“-Kanten. Bei der Darstellung werden nur die Knoten der zugehörigen Einfahrt und deren Nachfolger berücksichtigt.

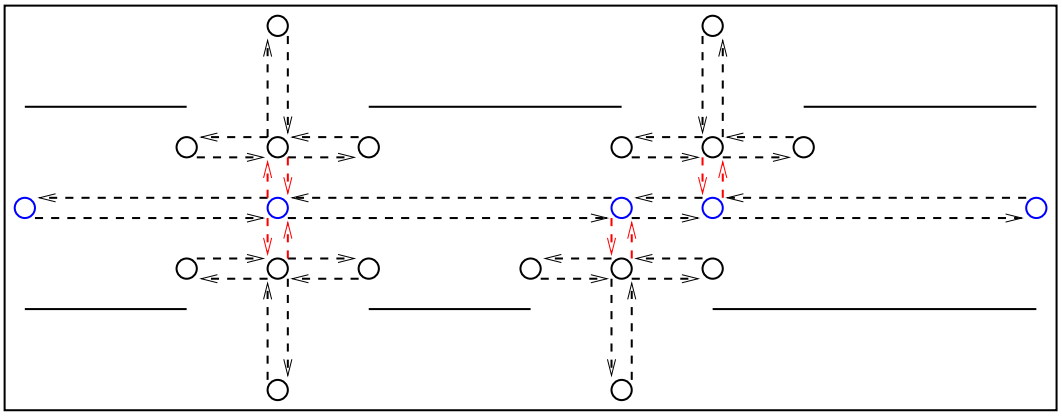

Abbildung 5.18: Korridor mit Graph und dazugehörigen Graphen der Einfahrten (original Korridorknoten, neue „`Lambda`“-Kanten)

Verbindung von Korridorgraphen über Kreuzungsgraphen

Die Knoten einer Kreuzung beinhalten als Untermodell den dazu gehörenden Korridor. Bei der Verbindung des Kreuzungsgraphen mit den passenden Korridorgraphen gibt es zwei Fälle:

1. Der Korridor ist zweimal im Kreuzungsgraphen als Untermodell vorhanden
2. Der Korridor ist einmal im Kreuzungsgraphen als Untermodell vorhanden

Im ersten Fall handelt es sich um einen durchgehenden und im zweiten Fall um einen endenden Korridor. Je nach Fall wird unterschiedlich vorgegangen. Die Vorgehensweisen werden im Folgenden detailliert beschrieben.

Durchgehender Korridor In diesem Fall wird als erstes eine Linie zwischen den beiden Knoten erzeugt. Mit Hilfe dieser Linie wird der Knoten im Korridorgraphen gesucht, der zwischen den beiden Punkten liegt. Dies geschieht mit Hilfe der Entfernung der Knoten zur Linie. Da auch die Öffnung zu einem anderen Korridor eine Einfahrt ist, besitzt der Korridor bei der Kreuzung auch einen Knoten. Dieser so gefundene Knoten dient als Ausgangspunkt für die Auswahl von zwei weiteren Knoten. Hierbei interessieren die direkten Nachfolger des Knotens. Abbildung 5.19 zeigt den Knoten zwischen den Kreuzungsknoten und dessen direkte Nachfolger.

Im nächsten Schritt werden die beiden Knoten der Kreuzung zwischen dem Mittel- und Nachfolgeknoten eingefügt. Hierfür wird als erstes ermittelt, welcher Kreuzungsknoten K_1 oder K_2 zwischen dem Mittelpunkt (M) und dem Nachfolger (N_1) liegt. Dies geschieht über die Orientierung der Linien von M nach N_1 und M nach K_1 beziehungsweise K_2. In Abbildung 5.19 liegt K_1 zwischen M und N_1. Vom Knoten M zum Knoten K_1 wird als erstes eine Kante mit der Bezeichnung „`Fahre geradeaus.`“ eingefügt. Für die Kante

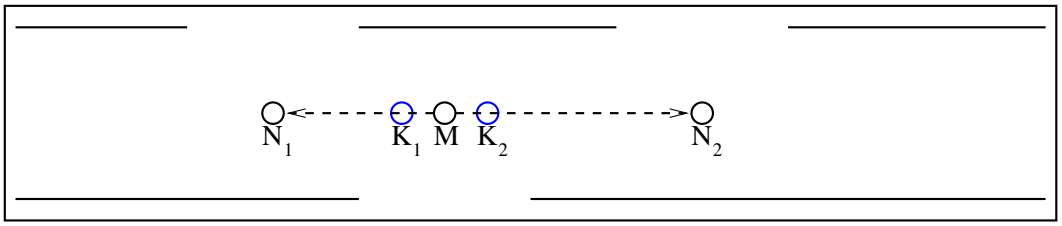

Abbildung 5.19: Korridor mit Mittelknoten M, Nachfolgerknoten N_1 und N_2 sowie die passenden Kreuzungsknoten K_1 und K_2

von K_1 nach M wird die Bezeichnung der im Korridorgraphen existierenden Kante von N_1 nach M benutzt. Als nächstes wird die Kante von N_1 nach K_1 erstellt. „`Fahre zur 1. Kreuzung.`“ lautet die Bezeichnung für diese Kante. Jetzt fehlt noch die Kante von K_1 nach N_1. Die Bezeichnung liefert die Kante von M nach N_1 aus dem Korridorgraphen. Als letztes werden die Kanten M nach N_1 und N_1 nach M gelöscht, da sie das Ergebnis verfälschen würden. Für die Knoten M, K_2 und N_2 wird entsprechend verfahren. Das Löschen der alten Kanten verhindert, dass es von N_1 nach N_2 eine direkte Verbindung über M gibt und damit die Kreuzung in den Bezeichnungen der Kanten nicht auftaucht. Abbildung 5.20 zeigt den neuen Graphen mit Blick auf einen Korridor. In der Darstellung sind auch alle Knoten des Kreuzungsgraphen angegeben.

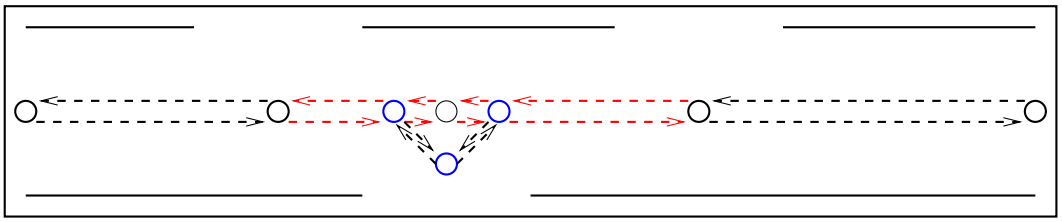

Abbildung 5.20: Korridor mit neuem Graphen, den Kreuzungsknoten und den neuen Kanten:
$M \to K_1$, $M \to K_2$ „`Fahre geradeaus.`“,
$K_1 \to M$, $K_1 \to N_1$ „`Fahre zur 1. Einfahrt rechts.`“,
$K_2 \to M$, $K_2 \to N_2$ „`Fahre zur 1. Einfahrt links.`“,
$N_1 \to K_1$, $N_2 \to K_2$ „`Fahre zur 1. Kreuzung.`“

Endender Korridor In diesem Fall wird der einzelne Knoten mit dem Endknoten des Korridors verbunden, der am nächsten an der Kreuzung liegt. Die Kante vom Kreuzungsknoten zum Endknoten bekommt die Bezeichnung „`Lambda`“. Die entgegengesetzte Kante erhält „`Fahre zur 1. Kreuzung.`“ als Bezeichnung. Abbildung 5.21 zeigt einen endenden Korridor mit der Verbindung zum Kreuzungsgraphen.

Abbildung 5.22 zeigt (1) zwei durchgehende Korridore verbunden durch den Kreuzungsgraphen und (2) den Graphen eines durchgehenden und eines endenden Korridors und deren Verbindung über den Kreuzungsgraphen.

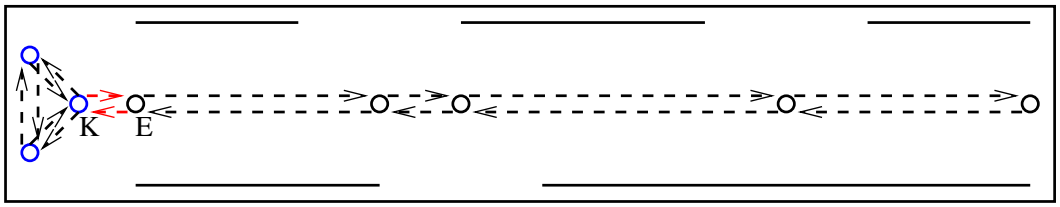

Abbildung 5.21: Korridor mit neuem Graphen, den Kreuzungsknoten und den neuen Kanten:
$E \rightarrow K$ „`Fahre zur 1. Kreuzung.`“,
$K \rightarrow E$ „`Lambda`“

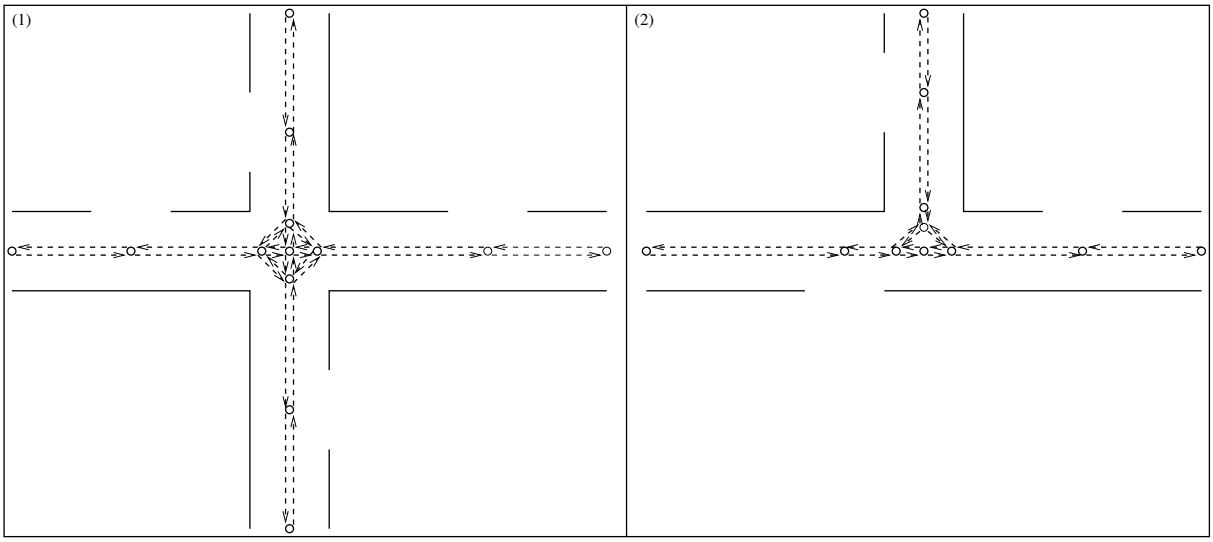

Abbildung 5.22: Komplette Graphen der Kreuzung und den zugehörigen Korridoren; (1) Von zwei durchgehenden Korridoren; (2) Von einem durchgehenden und einem endenden Korridor

5.4 Erzeugen der Roboterinstruktionen

In diesem Abschnitt wird beschrieben, wie die endgültigen Roboterinstruktionen gefunden werden. Die Erzeugung findet in zwei Schritten statt. Zuerst werden anhand eines geplanten Pfades die einzelnen Roboterinstruktionen gewonnen. Danach wird eine Zusammenfassung der einzelnen Instruktionen vorgenommen.

5.4.1 Die einzelnen Schritte

Die Pfadplanung erfolgt über eine Standard-Graphen Suche mit Start- und Zielpunkt als Eingabeparameter.

Abbildung 5.23 zeigt einen mit Hilfe der Suche gefundenen Beispielpfad. Entlang des Pfa-

des liefern die Kanten die Schritte, die den Roboter leiten sollen. Im vorherigen Abschnitt 5.3 wurde beschrieben, wie die Kanten erzeugt werden. Jede Kante hat dort eine Bezeichnung erhalten, die Aufschluss über die Aktion gibt, um von einem Knoten zum Nachfolger zu kommen. Diese Bezeichnung gibt auch den Schritt an, den der Roboter ausführen muss. Um nun die Schritte für die Abarbeitung des Pfades zu bekommen, brauchen nur die Bezeichnungen der Kanten gesammelt werden. Die Schritte für den Beispielpfad lauten:

```
 1) Startschritt.
 2) Fahre durch die Einfahrt.
 3) Lambda
 4) Fahre zur 1. Einfahrt rechts.
 5) Fahre zur 1. Kreuzung.
 6) Fahre nach links.
 7) Fahre zur 1. Kreuzung.
 8) Fahre zur 1. Einfahrt links.
 9) Fahre geradeaus.
10) Fahre zur 1. Einfahrt links.
11) Fahre zur 1. Kreuzung.
12) Fahre nach rechts.
13) Lambda
14) Fahre zur 1. Einfahrt rechts/links.
15) Lambda
16) Fahre durch die Einfahrt.
17) Zielschritt
```

Innerhalb dieser Schritte tauchen immer wieder „`Lambda`“-Anweisungen auf, die keine Anweisung enthalten. An diesen Stellen ist es häufig nötig, dem Roboter eine neue Richtung zuzuweisen. Zum Beispiel ist der Roboter nach dem 2. Schritt gerade aus einer Einfahrt gekommen und soll dann zur nächsten Einfahrt rechts fahren. Diese Einfahrt liegt auf der rechten Seite des Roboters in Fahrtrichtung. Da der Roboter aber senkrecht zum Verlauf des Korridors steht, muss ihm noch mitgeteilt werden, in welche Richtung er sich wenden soll. Diese Bearbeitung und die Zusammenfassung mehrerer Schritte zu einer Anweisung wird genauer im folgenden Abschnitt beschrieben.

5.4.2 Überarbeitung der einzelnen Schritte

Um eine Bearbeitung der Schritte vornehmen zu können, werden nicht nur die Instruktionen benötigt, sondern auch die Knoten, die durch die jeweilige Kante verbunden werden. Mit Hilfe dieser Informationen werden die im Folgenden beschriebenen Schritte vorgenommen.

„`Startschritt`“ und „`Zielschritt`“

Beim „`Startschritt.`“ handelt es sich um die Ausrichtung des Roboters auf den ersten Knoten eines Modells. Hierzu wird die Ausrichtung des Roboters mit der Ausrichtung der Linie von Startpunkt zum ersten Punkt des Graphen verglichen. Liegen beide Ausrichtungen im gleichen Winkelbereich, schaut der Roboter bereits in Richtung des ersten Modells. Sollten die Winkel zu weit voneinander abweichen, muss der Roboter sich drehen bis die Ausrichtung in etwa gleich ist. Je nach Abweichung der Ausrichtungen bekommt der Roboter als Anweisung „`Drehe dich nach rechts.`“, „`Drehe dich nach links.`“ oder „`Drehe dich um.`“ geliefert. Bei den ersten beiden Anweisungen soll sich der Roboter um 90° in die angegebene Richtung drehen. Beim Umdrehen soll sich der Roboter um 180° drehen.

Beim „`Zielschritt.`“ braucht dem Roboter nur mitgeteilt werden, dass er sein Ziel erreicht hat.

Umwandlung von „`Lambda`“-Kanten

Wird ein „`Lambda`“-Schritt erreicht, muss als erstes überprüft werden, ob vor oder nach dem „`Lambda`“ nicht vielleicht schon ein Richtungsschritt erfolgt. Im oberen Beispiel sieht man diesen Fall zum Beispiel bei Schritt 13. Schritt 14 gibt die Anweisung, dass der Roboter nach rechts fahren soll. Damit steht der Roboter schon in richtiger Richtung und dieses „`Lambda`“ braucht nicht mehr beachtet zu werden. Anders sieht es bei Schritt 3 und Schritt 15 aus. Hier muss der Roboter wieder eine Dreh-Anweisung bekommen. Zur Bestimmung der Richtung, in die sich der Roboter drehen soll, gibt es zwei Möglichkeiten. Als erstes kann die Bezeichnung der vorangehenden Kante untersucht werden. Lautet die Bezeichnung der Kante zum Beispiel „`Fahre zur 1. Einfahrt links.`“ befindet der Roboter sich vor einer linken Einfahrt. Wenn eine „`Lambda`“-Kante auf eine solche Kante folgt, kann Aufgrund des Aufbaus des Graphen davon ausgegangen werden, dass sich der Roboter nach *links* drehen muss. Hat die vorherige Kante keine eindeutige Richtung im Bezeichner, wird die Ausrichtung der „`Lambda`“-Kante benutzt, um die Drehrichtung des Roboters zu bestimmen. Es wird ermittelt, ob der nächste Punkt rechts oder links von dieser Kante liegt. Abbildung 5.24 zeigt Beispiele für die Bestimmung der Drehrichtung an einer „`Lambda`“-Kante.

Zusammenfassung von einzelnen Schritten

Hierbei werden Schritte zwischen zwei Richtungsänderungen zusammengefasst. Um die Art der Zusammenfassung zu bestimmen, wird der letzte Schritt vor der Richtungsänderung bestimmt. Anschließend werden die Kanten mit gleiche Bezeichnung gezählt und zusammengefasst. Lautet der letzte Schritt vor einer Richtungsänderung zum Beispiel „`Fahre zur 1. Einfahrt links.`“ werden die Kanten zwischen der letzten und der aktuellen Richtungsänderung mit gleichem Bezeichner gesucht und gezählt. Außerdem werden Kanten mitgezählt, bei denen die Bezeichnung „`Fahre zur 1. Einfahrt rechts/links.`“

lautet. Bei einer Kante mit dieser Bezeichnung gibt es am Ziel auf beiden Seiten eine Einfahrt, muss daher beachtet werden. Finden sich hierbei zum Beispiel drei Kanten, bekommt der Roboter die Anweisung „`Fahre zur 3. Einfahrt links.`“. Das gleiche gilt auch für den Schritt „`Fahre zur 1. Einfahrt rechts.`“. Lautet die letzte Anweisung vor dem Richtungswechsel „`Fahre zur 1. Einfahrt rechts/links.`“, wird mit Hilfe der Richtungsanweisung die Seite bestimmt, die gezählt werden muss. Die letzte Anweisung, die gezählt werden kann, ist „`Fahre zur 1. Kreuzung.`“. Hierbei wird ebenso verfahren wie bei den Einfahrten.

Aus dem oben gegebenen Beispiel ergibt sich dann folgende Befehlsfolge, wobei davon ausgegangen wird, dass der Roboter am Startpunkt schon in die richtige Richtung gedreht ist.

```
1) Fahre durch die Einfahrt.
2) Drehe dich nach rechts.
3) Fahre zur 1. Kreuzung.
4) Fahre nach links.
5) Fahre zur 2. Kreuzung.
6) Fahre nach rechts.
7) Fahre zur 1. Einfahrt links.
8) Drehe dich nach links.
9) Ziel erreicht.
```

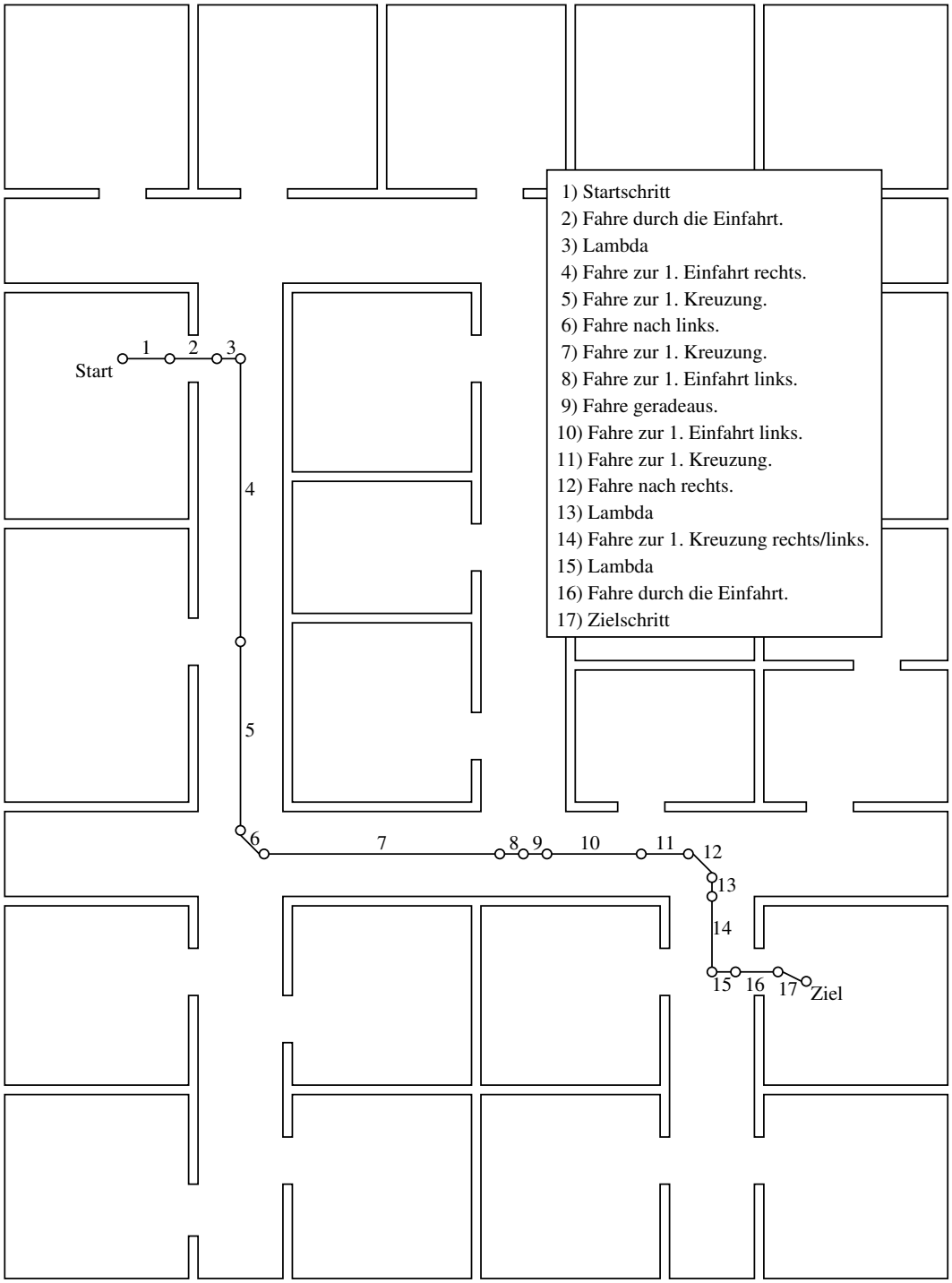

Abbildung 5.23: Beispiel eines Pfades mit 17 Schritten.

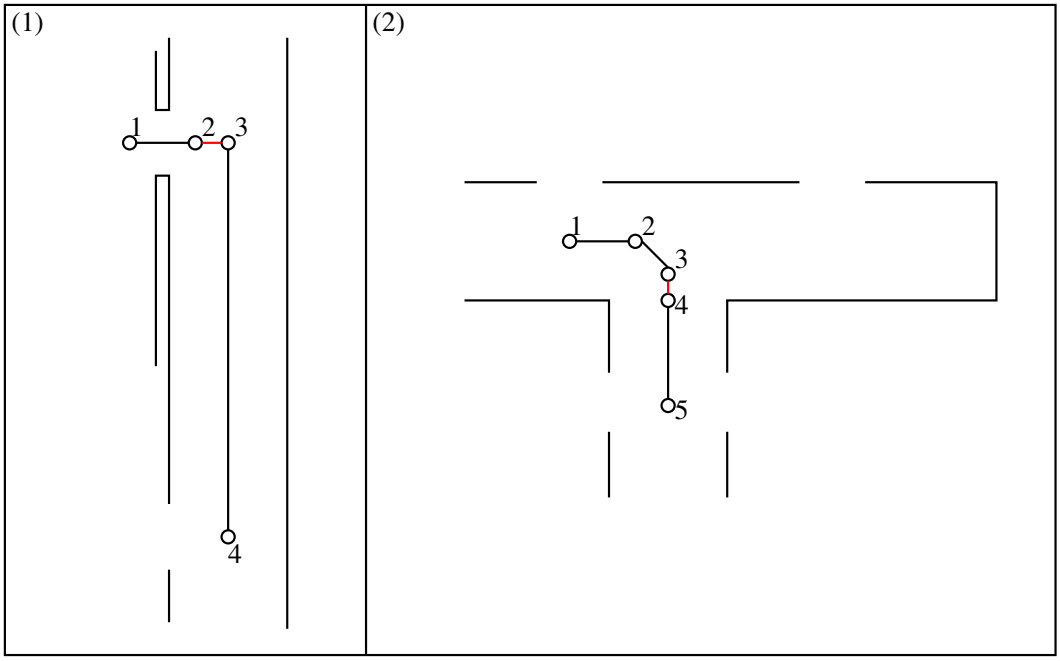

Abbildung 5.24: Lambdaersetzung: `Lambda-Kante`;
(1) 1 → 2 „`Fahre durch die Einfahrt.`“, 3 → 4 „`Fahre zur 1. Einfahrt rechts.`“; Punkt 4 liegt rechts von der Kante 2 → 3, daher ergibt sich die Anweisung „`Drehe dich nach rechts.`“
(2) 2 → 3 „`Fahre nach rechts.`“, daher wird die „`Lambda`“-Kante einfach gestrichen.

Kapitel 6

Evaluation der Arbeit

In diesem Kapitel werden Tests des entwickelten Verfahrens durchgeführt. Als erstes wird die Erkennung und Instruktionsgenerierung auf gegebenen Karten getestet. Anschließend wird die Erkennung auf reale Roboterdaten angewandt, um die prinzipielle Eignung für den Umgang mit realen Sensordaten eines Roboters zu evaluieren.

6.1 Evaluation der Erkennung und Instruktionsgenerierung auf Karten

6.1.1 Evaluation der Erkennung

Im Folgenden basierend auf der in Anhang C dargestellten Karte beispielhaft die Erkennung und die Erzeugung der Graphen dargestellt. In Abschnitt 5.4 wurde basierend auf dieser Karte ein Pfad geplant und die Roboterinstruktionen erzeugt. Auf die dort dargestellten Ergebnisse wird im Anschluss an Evaluation der Erkennung und der Erzeugung des Graphen noch einmal eingegangen.

Als erstes werden die Wände dieser Karte erkannt. Hierbei sollten in der Testkarte fast alle Linien zu einer Wand werden. Kurze Linien, wie sie als Beispiel in Abbildung 6.1 dargestellt sind, sollten dabei nicht zu einer Wand werden. Den Erwartungen entsprechend wurden alle Wände erkannt.

Nachdem alle Wände erkannt worden sind, sollen Einfahrten und einfache Korridore erkannt werden. Abbildung 6.2 zeigt beispielhaft zwei erkannte Einfahrten. Da die Einfahrten auf Wänden basieren, zeigt die Abbildung die Wände der Einfahrten. Diese wurden vorher erfolgreich erkannt.

Nachdem die Einfahrten und einfachen Korridore erkannt wurden, sollen als nächstes die langen Korridore erzeugt werden. Abbildung C.2 zeigt rot die zu erkennenden langen Korridore. Abbildung 6.3 zeigt beispielhaft den langen Korridor 1 der Testkarte. Dieser

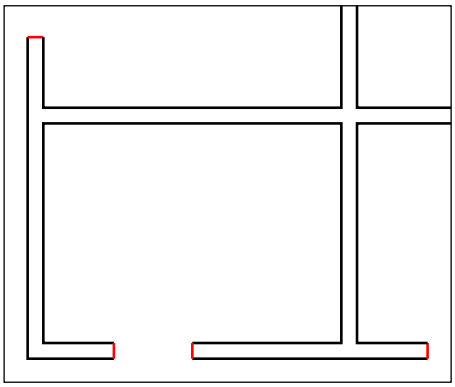

Abbildung 6.1: Die roten Linien werden keine Wände, da sie zu kurz ist. Alle anderen Linien werden zu je einer Wand.

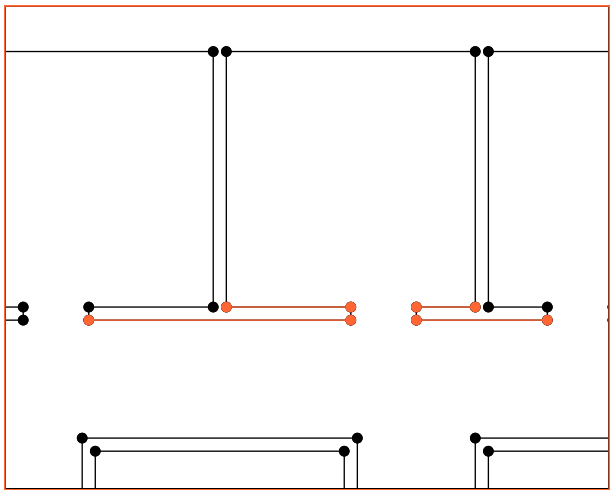

Abbildung 6.2: Zwei erkannte Einfahrten. Die erste Einfahrt besteht aus den beiden oberen Linien während die zweite Einfahrt aus den unteren Linien besteht. Jede Linie für sich, stellt eine Wand dar.

lange Korridor wäre nicht erkannt worden, wenn nicht auch die Erkennung der einfachen Korridore funktioniert hätte.

Als letztes sollen, mit Hilfe der erkannten langen Korridore, Kreuzungen erkannt werden. In Abbildung C.2 sind die zu erkennenden fünf Kreuzungen blau markiert. Mit Hilfe der zugehörigen Korridore wurden die entsprechenden Kreuzungen erkannt.

6.1.2 Evaluation der Erzeugung des Graphen

Nach der Erkennung der Modelle werden Teilgraphen erzeugt. Als erstes werden für erkannte Einfahrten, Korridore und Kreuzungen die Teilgraphen erzeugt. Für Wände, die kein Bestandteil eines langen Korridors sind, werden im Anschluss Teilgraphen erzeugt.

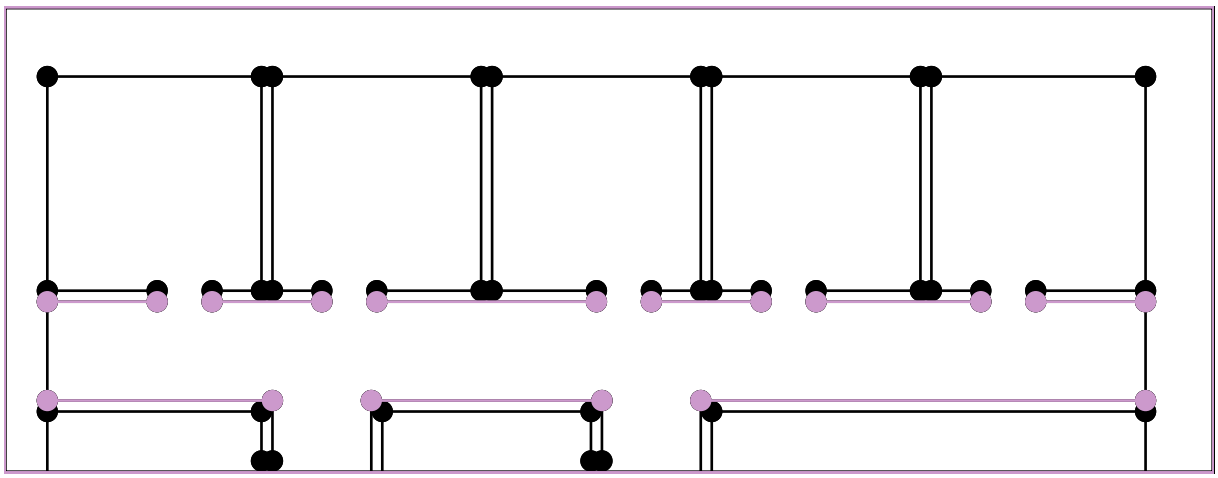

Abbildung 6.3: Ein langer Korridor, bestehend aus fünf Einfahrten im oberen Teil und zwei Einfahrten im unteren Teil. Mit Hilfe von mehreren einfachen Korridoren wurde der lange Korridor zusammengesetzt.

Anhang D zeigt den so erzeugten Graphen. Im folgenden werden beispielhaft erzeugte Teilgraphen und deren Verbindungen dargestellt. Abbildung 6.4 zeigt die erzeugten Graphen der in Abbildung 6.2 erkannten Einfahrten. Abbildung 6.5 zeigt den Graphen des in Abbildung 6.3 dargestellten Korridors.

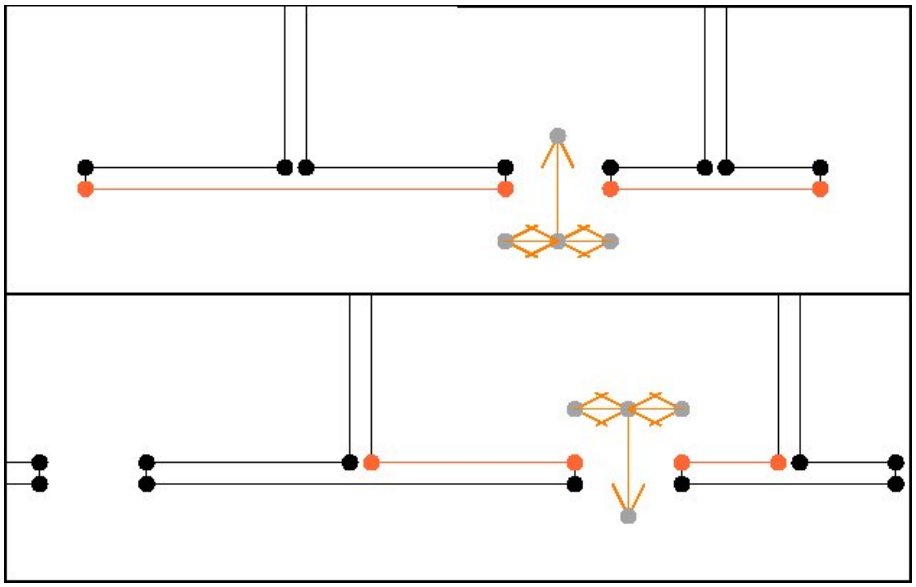

Abbildung 6.4: Die Graphen der in Abbildung 6.2 erkannten Einfahrten

Im nächsten Schritt werden Teilgraphen zusammengefügt. Als erstes werden die Graphen von nahe beieinander liegenden Einfahrten verbunden. Die Knoten der in Abbildung 6.4 dargestellten Graphen liegen dicht genug beieinander, so dass der Graph in Abbildung 6.6 erzeugt wird.

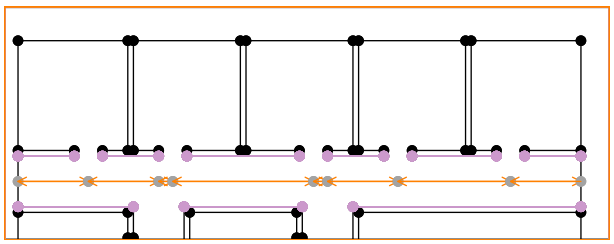

Abbildung 6.5: Der Teilgraph, der für den in Abbildung 6.3 erkannten Korridor erzeugt wurde.

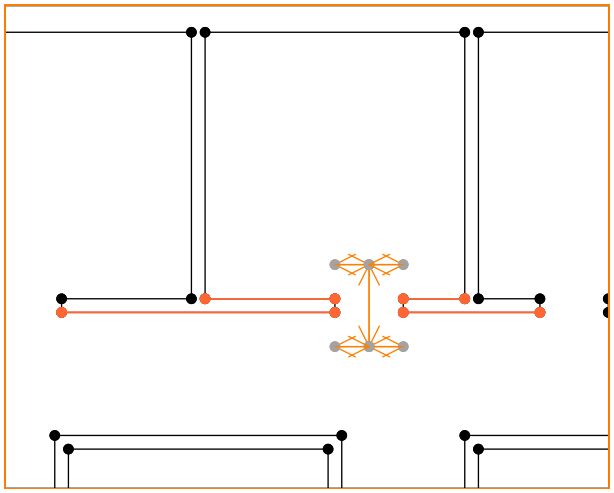

Abbildung 6.6: Fusion der beiden in Abbildung 6.4 dargestellten Graphen zu einem größeren Teilgraphen.

Der nächste Schritt verbindet die Graphen der Korridore mit den Graphen der im Korridor enthaltenden Einfahrten. Hieraus ergibt sich in der Evaluation der in Abbildung 6.7 gezeigte Graph. Bei der Abbildung wurde auf die Darstellung aller Graphen der Einfahrten verzichtet und nur der Graph der beiden im Beispiel gezeigten Einfahrten eingefügt. Jeder senkrechte Pfeil vom Korridorgraphen weg stellt einen Verbindung zu einem Graphen einer Einfahrt dar.

Der in Anhang D dargestellte Graph wurde für die Erzeugung der Roboterinstruktionen, welche in Abbildung 5.23 dargestellt sind, gefunden. Die in Abschnitt 5.4 erzeugten Instruktionen und deren Zusammenfassung entsprach dem erwarteten Ergebnis.

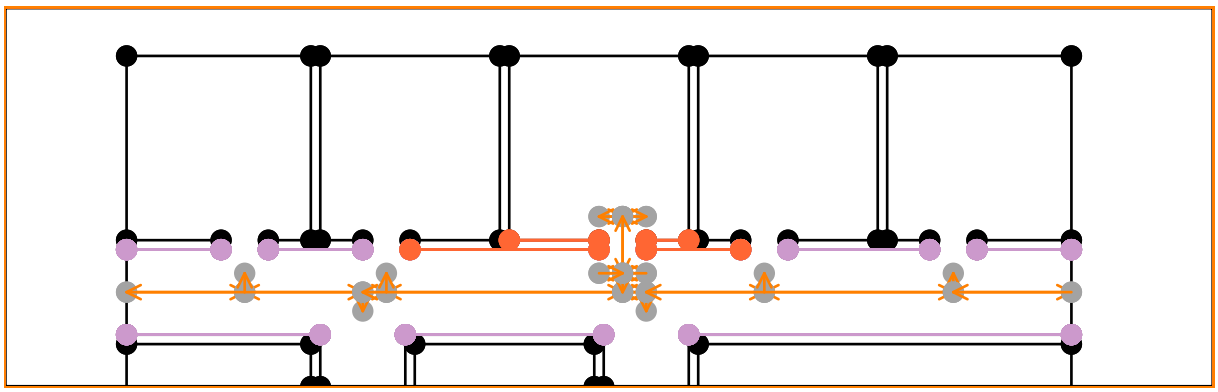

Abbildung 6.7: Korridorgraph mit senkrechten Pfeilen, die je eine Verbindung zu einem Einfahrtsgraphen darstellen. Zusätzlich wurde der Graph aus Abbildung 6.6 in diese Darstellung eingefügt.

6.2 Evaluation der Erkennung auf realen Sensordaten

Für diese Evaluation habe ich eine Karte der 5. Ebene des MZH an der Uni Bremen an das Programm übergeben. Dieser Test soll überprüfen, ob die Erkennung der Modelle wie in Kapitel 4 beschrieben auch auf Sensordaten funktioniert. Der für diesen Test benutzte Roboter ist ein Pionier 2. Als Sensor besitzt dieser Roboter einen Laserscanner mit 180° Sichtfeld. Die Auflösung dieses Sensors beträgt 361 Scanpunkte. Anhang B.1 zeigt die Karte dieser Ebene mit den Punkten, an denen die folgenden Aufnahmen entstanden sind. In diesem Experiment wird eine Instruierung des Roboters simuliert, die ihn von Raum 5300 nach Raum 5280 führt. Die hierfür verwendeten Instruktionsschritte lauten:

```
Fahre durch die Einfahrt.
Fahre zur 1. Einfahrt links.
Drehe dich nach links.
Fahre durch die Einfahrt.
```

Abbildung 6.8 zeigt den Startpunkt der Testfahrt. Die Abbildung zeigt die Sicht eines Menschen an diesem Punkt. Zusätzlich sind dunkel hinterlegt die Scandaten des Roboters zu sehen. Für die Erkennung werden Linien benötigt, welche aus den Punkten bereits extrahiert wurden. Hierfür wurde ein Verfahren zur Extraktion von polygonalen Kurven aus Sensordaten verwendet [7]. Die polygonalen Kurven werden anschließend in einzelne Linienprüfsegmente gespalten.

Abbildung 6.9 zeigt die mit Hilfe der extrahierten Linien erkannten Modelle. In der Abbildung sieht man die erkannten Wände (1) und die erkannten Einfahrten (2). Da der Roboter eine Einfahrt direkt vor sich sehen kann, kann er die Instruktion „`Fahre durch die Einfahrt.`“ ausführen.

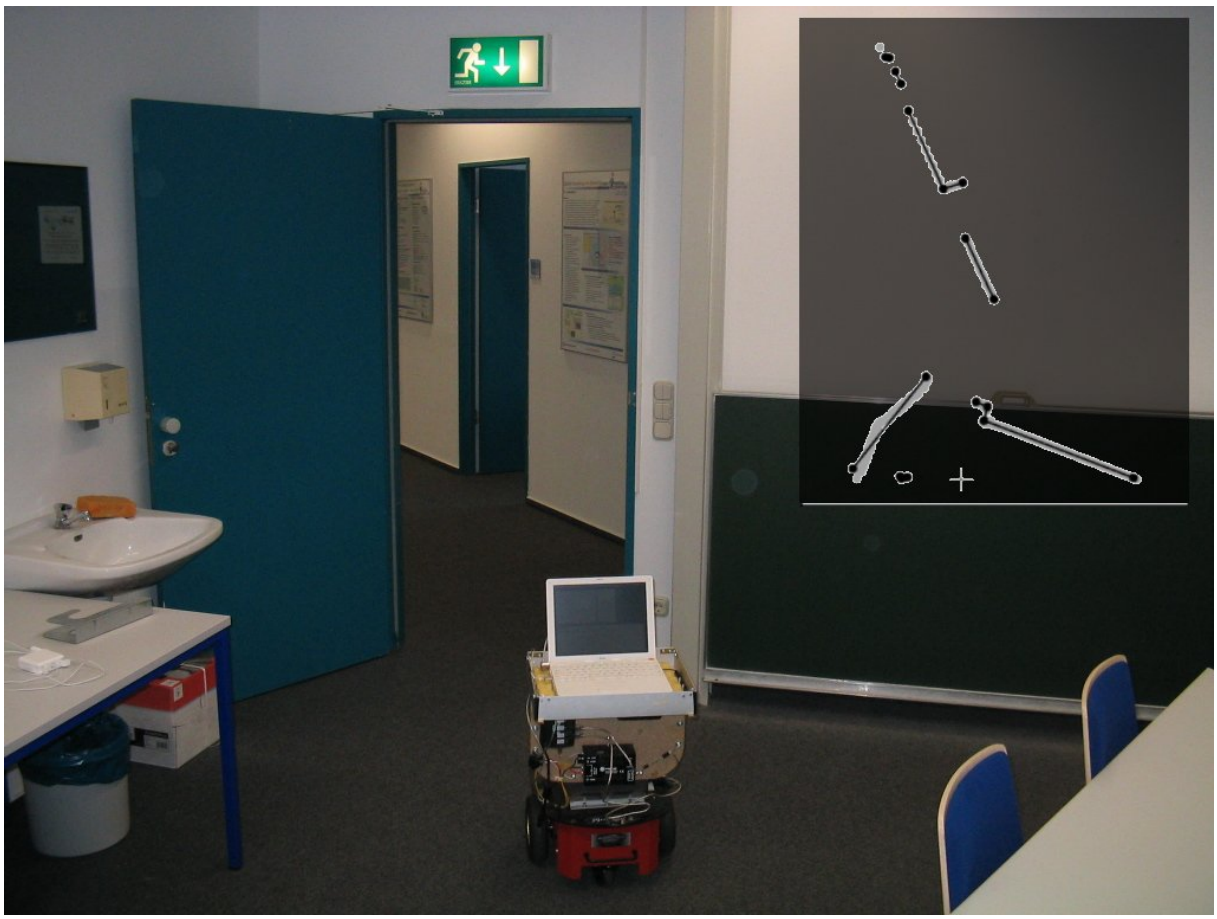

Abbildung 6.8: Sichtbarkeit aus einem Raum in einen Flur. Im dunklen Rechteck sind die Sichtbarkeitsdaten des Roboters zu sehen. Aus den Scanpunkten (grau) wurden bereits Linien extrahiert.

Nach der Durchfahrt befindet sich der Roboter an der zweiten Position. Abbildung 6.10 zeigt das Sichtfeld des Roboters an dieser Position.

Mit Hilfe der aus dem Scan extrahierten Linien können erneut Modelle erkannt werden. Abbildung 6.11 zeigt die erkannten Wände (1), Einfahrten (2) und einen langen Korridor (3).

Nach ca. 4m Wegstrecke entlang des Korridors erreicht der Roboter die dritte Position, an dem er die Einfahrt sehen kann, die zu dem Raum führt, in den der Roboter fahren soll. Abbildung 6.12 zeigt diese Position mit dem korrespondierenden Scan.

Die gesuchte Einfahrt wird auf den extrahierten Daten gefunden. Das Ziel des Roboters im Korridor war die erste Einfahrt auf der linken Seite. Abbildung 6.13 zeigt, das die Einfahrt bereits erkannt werden kann. Außerdem sind in der Abbildung die erkannten Wände und Korridore dargestellt.

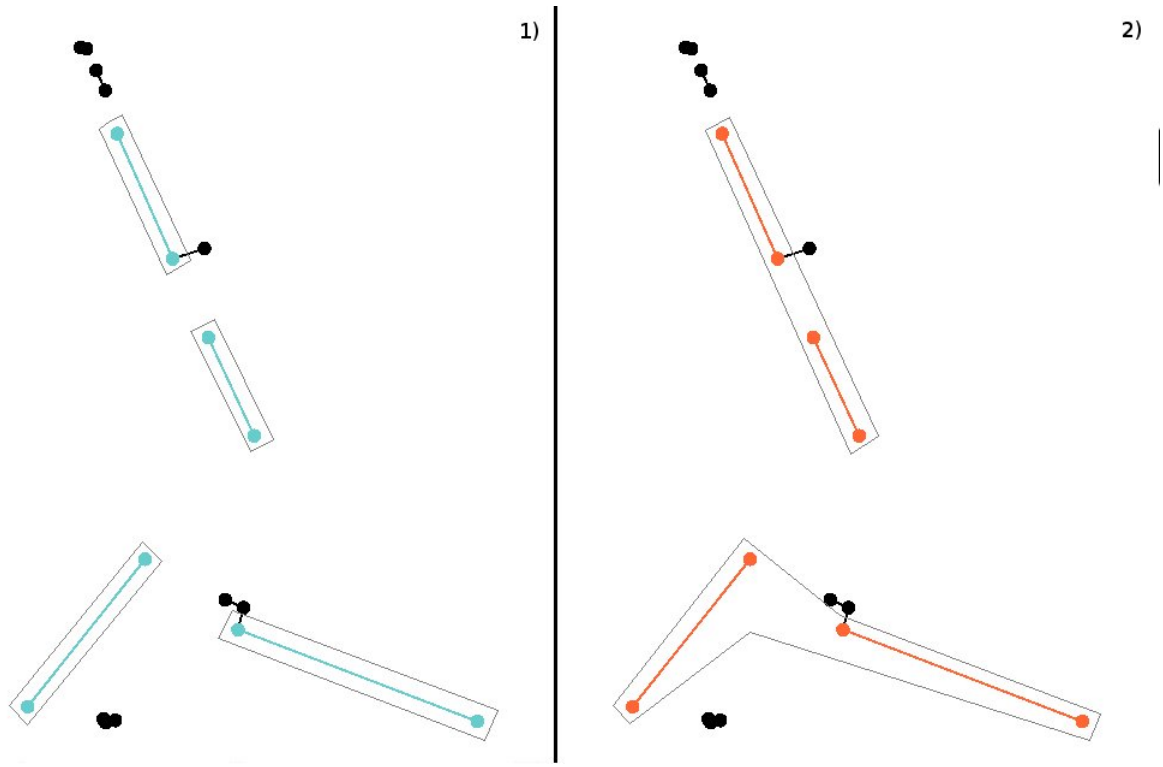

Abbildung 6.9: (1) Erkannte Wände aus den in Abbildung 6.8 extrahierten Linien. (2) Mit den Wänden erkannte Einfahrten.

Der letzte Punkt, bevor der Roboter sein Ziel erreicht, liegt vor der Einfahrt. Hierzu hat sich der Roboter bereits vor der Einfahrt nach links gedreht. Abbildung 6.14 zeigt diese Position.

Um seinen letzten Befehl („`Fahre durch die Einfahrt.`“) ausführen zu können, muss der Roboter eine Einfahrt erkennen. Abbildung 6.14 zeigt, dass eine Einfahrt erkannt wird. Damit kann der Roboter seinen letzten Schritt machen und ist am Ziel.

Bei diesem Experiment musste ein Parameter für die Erkennung der Einfahrt verändert werden. In Abbildung 6.8 ist eine die Tür des Raumes zu sehen. Der Winkel, den die Tür zu der Wand rechts neben der Türöffnung bildet, hätte sonst eine Erzeugung der Einfahrt verhindert. Hieran sieht man einen Unterschied zwischen der Karte, auf der die Modelle gesucht werden, und der realen Umgebung. Um diese Problem zu umgehen müsste man Einfahrten mit Türen einführen. Wobei in diesem Fall die Türen irgendwie an der Einfahrt verankert sein könnten. Im Rahmen dieser Arbeit ist dieses Konzept nicht verfolgt worden, da eine Erkennung auf der Karte nicht möglich ist. In der Karte werden Öffnungen nicht unterschieden.

Abbildung 6.10: Sichtbarkeit nach Fahrt durch die erkannte Einfahrt. Im dunklen Rechteck sind die Sichtbarkeitsdaten des Roboter zu sehen. Aus den Scanpunkten (grau) wurden bereits Linien extrahiert.

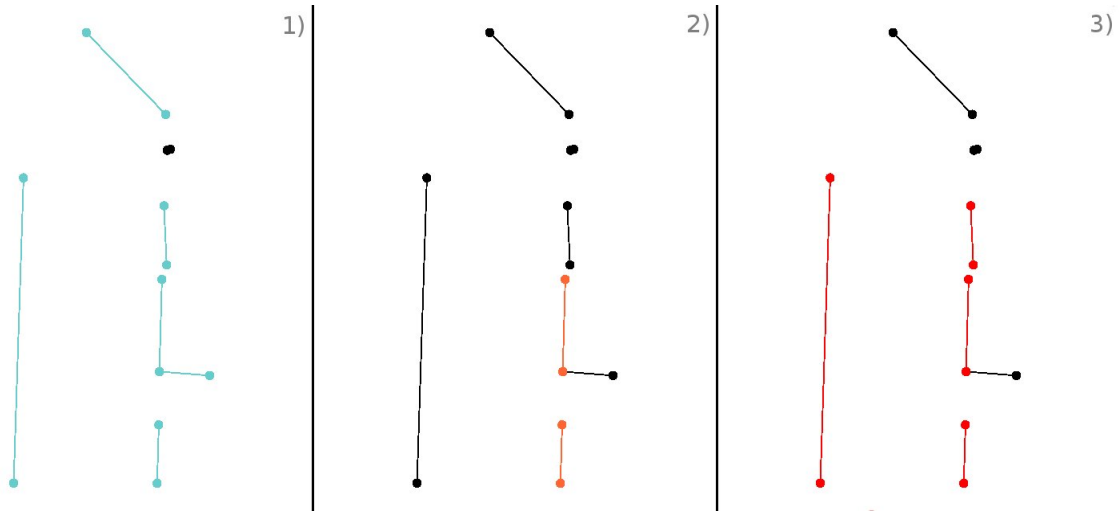

Abbildung 6.11: (1) Erkannte Wände aus den in Abbildung 6.10 extrahierten Linien. (2) Mit den Wänden erkannte Einfahrt. (3) Erkannter langer Korridor: eine Einfahrt (2) und drei einfache Korridore, wobei jeder Korridor aus der linken Wand und einer der drei rechten Wände besteht.

Abbildung 6.12: Sichtbarkeit auf dem Flur vor dem Zielraum. Im dunklen Rechteck sind die Sichtbarkeitsdaten des Roboter zu sehen. Aus den Scanpunkten (grau) wurden bereits Linien extrahiert.

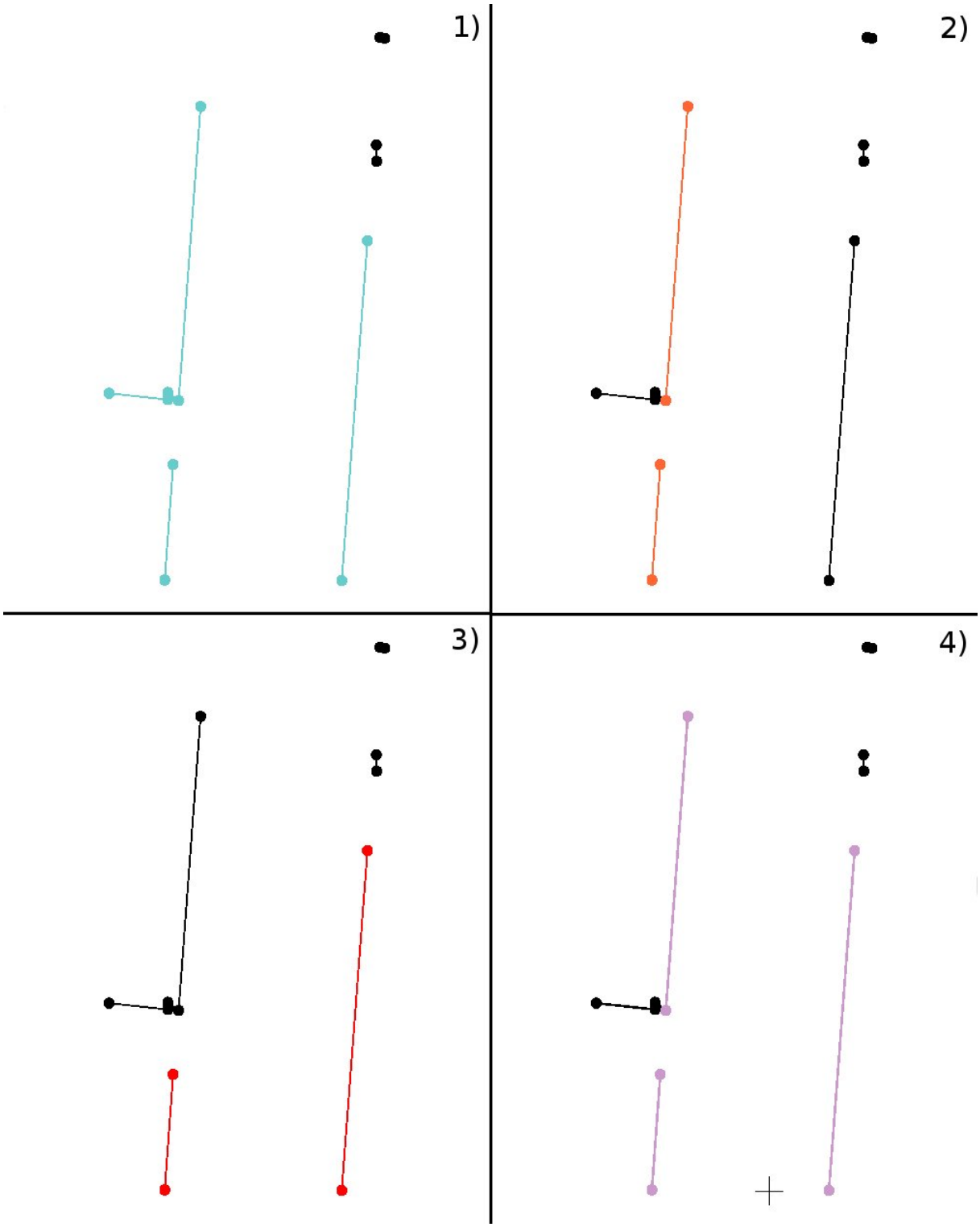

Abbildung 6.13: (1) Aus den in Abbildung 6.12 extrahierten Linien erkannte Wände. (2) Aus den Wänden erkannte Einfahrt. (3) Aus den Wänden erkannter Korridor. (4) Aus der Einfahrt und dem Korridor erkannter langer Korridor.

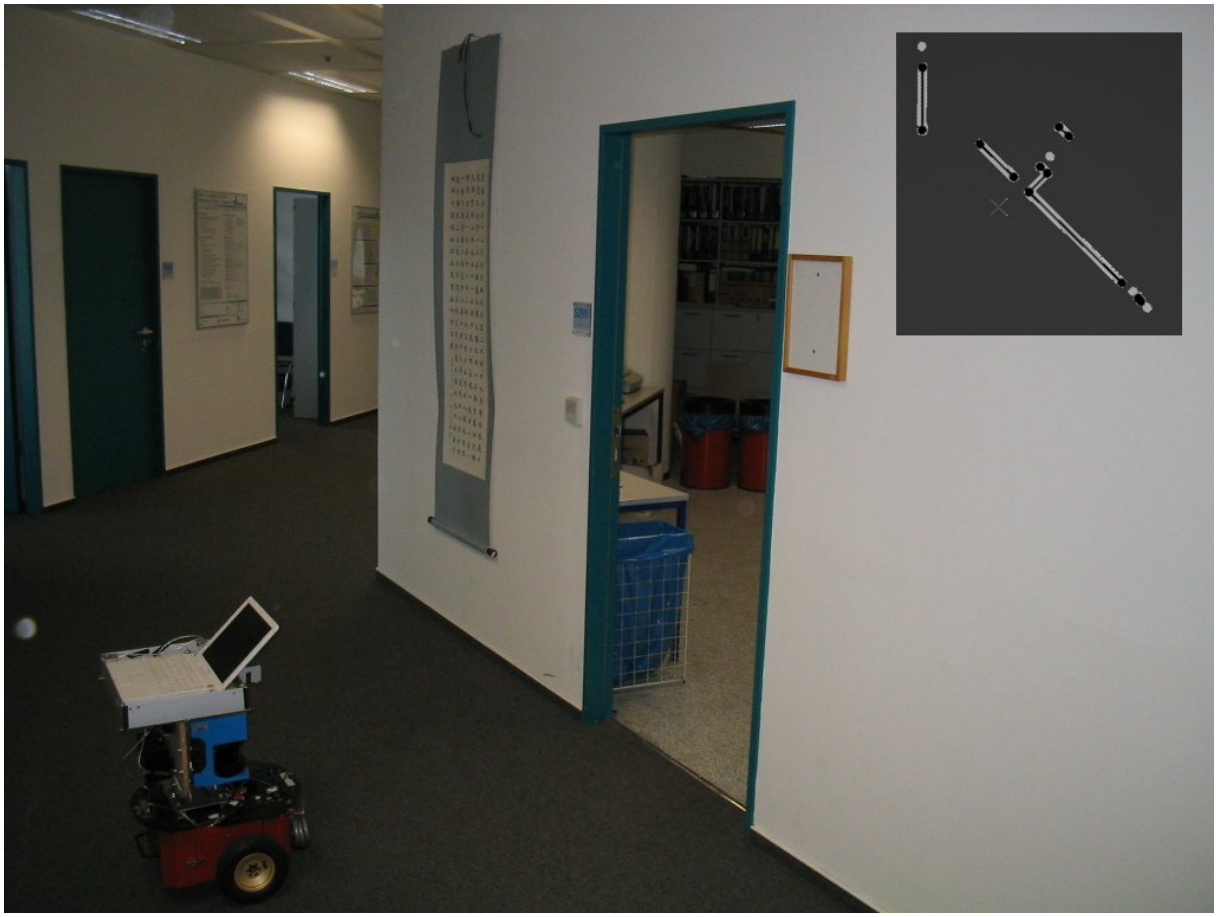

Abbildung 6.14: Sichtbarkeit auf die Einfahrt zum Zielraum. Im dunklen Rechteck sind die Sichtbarkeitsdaten des Roboters zu sehen. Aus den Scanpunkten (grau) wurden bereits Linien extrahiert.

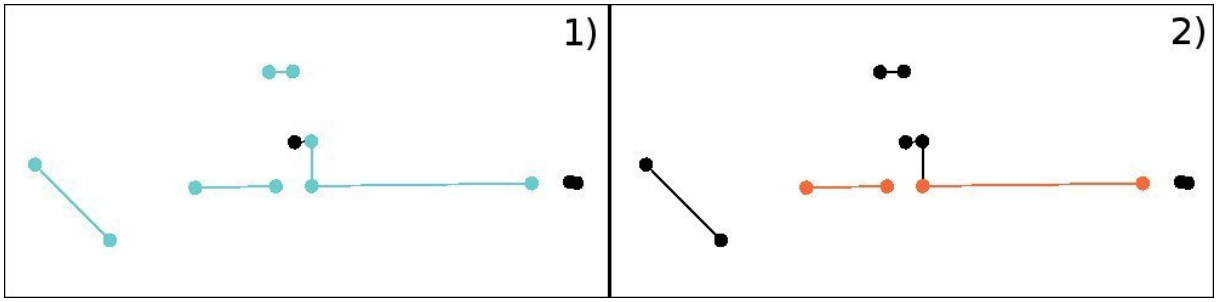

Abbildung 6.15: (1) Aus den in Abbildung 6.14 extrahierten Linien erkannte Wände. (2) Mit den Wänden erkannte Einfahrt.

Kapitel 7

Abschluss

In dieser Arbeit wurde ein Verfahren entwickelt, das einen Roboter mit Hilfe von natürlichsprachigen Instruktionen von seinem Startpunkt zu einem Ziel leitet. Die Instruktionen werden anhand erkannter Umgebungsmerkmale automatisch generiert. Die Umgebungsmerkmale werden vorher automatisch aus einer Umgebungskarte erkannt.

7.1 Zusammenfassung

In dieser Arbeit ist ein Verfahren entwickelt worden, mit dem auf einer Umgebungskarte Modelle erkannt werden und mit deren Hilfe anschließend Instruktionen für die Leitung eines Roboter erzeugt werden. Hierzu werden als erstes verschiedene Modelltypen aus der Karte extrahiert. Zunächst sind es Wände. Dies ist das einfachste in dieser Arbeit verwendete Modell. Eine Wand besteht aus einer Liste von Linien. Eine Wand wird durch die Nähe der Linien, deren Parallelität und deren Gesamtlänge beschrieben. Diese Beschreibung der Modelle wird mit Hilfe von Relationen vorgenommen. Als nächstes werden Einfahrten und Korridore erkannt. Der Erkennung dieser Modelle wird auf den zuvor erkannten Wänden vorgenommen. Eine Einfahrt besteht aus zwei Wänden, die in einem Entfernungsbereich zueinander liegen. Außerdem sollten beide Wände von einer Position vor der Einfahrt von vorne gesehen werden. Dazu wird die Ausrichtung der beiden Wände bestimmt. Damit aus zwei Wänden eine Einfahrt wird, darf zwischen den beiden Wänden kein Hindernis liegen. Auch ein einfacher Korridor besteht aus zwei Wänden, die einen bestimmten Abstand zueinander haben. Außerdem müssen die Wände sich gegenüber liegen. Ein Korridor wird dann erzeugt, wenn, von einem Punkt zwischen den Wänden, beide Wände gesehen werden können. Nach der Erkennung der Einfahrten und einfachen Korridore werden lange Korridore erzeugt. Diese Korridore erhalten zuerst einen einfachen Korridor. Anschließend werden Einfahrten und weitere einfache Korridore hinzugefügt, die auf einer Wand des langen Korridors basieren. Dieser Vorgang wird so lange wiederholt, bis kein weiterer Korridor und keine weitere Einfahrt hinzugefügt werden kann. Sind die langen Korridore erkannt, wird versucht mit ihnen Kreuzungen zu erkennen. Kreuzungen bestehen aus zwei langen Korridoren, die eine gewisse Nähe zu

einander haben. Die Nähe allein ist nicht ausreichend für die Erzeugung einer Kreuzung. Zusätzlich dürfen die Korridore nicht parallel zueinander liegen.

Nachdem alle Modelle auf der Karte erkannt sind, wird für jedes Modell ein Teilgraph erzeugt. Dieser Graph dient später für die Erzeugung von Roboterinstruktionen entlang eines Pfades. Dieser Pfad wird anhand einer Standard-Graphen-Suche bestimmt. Jedem Modell werden dazu eine Menge von Knoten zugewiesen. Diese Knoten werden durch Kanten verbunden. Jeder Knoten ist ein Dreitupel. Dieses Tupel besteht aus einem Punkt, der den Knoten in die Umgebungskarte einbettet, einem Modell, zu dessen Teilgraphen der Knoten gehört, und eine Menge von Untermodellen. Diese Untermodelle gibt es zum Beispiel bei Korridorgraphen, da die langen Korridore unter anderem aus Einfahrten bestehen. Die Kanten sind ein Tupel aus Roboterinstruktion und Länge der korrespondierenden Wegstrecke.

Als erstes werden die Teilgraphen der Einfahrten erzeugt. Die Teilgraphen der Einfahrten bilden die möglichen Aktionen ab, die ein Roboter an dem Modell ausführen kann. Die Aktionen an einer Einfahrt sind das Vorbeifahren und das Hindurchfahren. Als nächstes werden die Teilgraphen der langen Korridore erzeugt. Hier kann der Roboter bis zum Ende des Korridors oder bis zu einer Einfahrt fahren. Anschließend werden die Teilgraphen der Kreuzungen erzeugt. Hier kann der Roboter entweder geradeaus fahren oder nach rechts beziehungsweise links abbiegen. Zuletzt wird für jede Wand, die kein Untermodell eines langen Korridors ist, ein Teilgraph erzeugt. Für eine Wand ist die einzige Aktion das Entlangfahren.

Nachdem alle Teilgraphen erzeugt sind, werden diese Teilgraphen miteinander verbunden, um einen Gesamtgraphen zu erhalten, auf dem die Pfadplanung vorgenommen werden kann. Hierbei werden als erstes Einfahrtsgraphen miteinander verbunden. Es werden Einfahrtsgraphen verbunden, die direkt beieinander liegen. Auf diese Art und Weise werden zum Beispiel Verbindungen zwischen zwei Räumen im Graphen dargestellt. Anschließend werden Korridor- und Einfahrtsgraphen verbunden. Da jeder Knoten eines Korridorgraphen, der sich auf eine Einfahrt bezieht, diese als Untermodell liefert, kann mit der Hilfe der Einfahrt einfach der korrespondierende Einfahrtsgraph ausgewählt und verbunden werden. Im nächsten Schritt werden Korridorgraphen über die Kreuzungsgraphen verbunden. Auch hierbei helfen die Untermodelle in den Knoten der Kreuzungsgraphen, den richtigen Korridorgraphen auszuwählen. Als letztes müssen noch die Wandgraphen zum Gesamtgraphen hinzugefügt werden. Hierbei werden die Knoten mit Knoten anderer Graphen verbunden, deren Punkte am nächsten an dem zu verbindenden Knoten liegt.

Ist der Gesamtgraph fertiggestellt, kann die Pfadplanung vorgenommen werden. Entlang des gefundenen Pfades werden die Instruktionen aus den Kanten extrahiert und anschließend zusammengefasst. Hierbei werden `Lambda`-Anweisungen wenn nötig in Richtungsänderungen umgewandelt. `Lambda`-Kanten wurden eingefügt, wenn für eine Kante keine Roboterinstruktion genommen werden kann. Zum Beispiel für die Verbindung von Korridor- und Einfahrtsgraphen. Zusätzlich werden entlang einer Fahrtrichtung des Roboters gleiche Instruktionen zusammengefasst. Dies geschieht mit Hilfe der letzten Instruktion vor einer Richtungsänderung. Ist die letzte Instruktion zum Beispiel „`Fahre zur`

`1. Einfahrt rechts.`“, werden die gleichen Instruktionen gezählt und zum Beispiel zu „`Fahre zur 3. Einfahrt rechts.`“ zusammengefasst. Nach der Zusammenfassung der Instruktionen erhält man die Roboterinstruktionen, die den Roboter von seiner Startposition zur erwünschten Zielposition führt.

7.2 Bewertung der Arbeit als ganzes

Das Ziel dieser Arbeit ist es gewesen, ein Verfahren zu entwickeln, das einen Roboter innerhalb einer Büroumgebung von seiner aktuellen Position zu einer Zielposition führt. Dabei soll eine Sprache verwendet werden, die sich an Wegbeschreibungen für Menschen orientiert. Wegbeschreibungen für Menschen nutzen Landmarken. Aus diesem Grund sollen als erstes Umgebungsmerkmale aus einer Karte extrahiert werden. Anhand dieser Umgebungsmerkmale sollen dann Roboterinstruktionen generiert werden.

Mit dem hier vorgestellten Verfahren können Wegbeschreibungen generiert werden. Diese Wegbeschreibungen müssen anschließend noch von eingesetzten Robotern interpretiert werden, was nicht zu dieser Arbeit gehörte. Die Karten, auf denen die Umgebungsmerkmale extrahiert wurden, beinhalten keine Hindernisse wie zum Beispiel Stühle. Hier müsste bei der Erkennung des Roboters eventuell noch etwas verändert werden. Außerdem zeigte sich in der Evaluation mit realen Sensordaten, dass zum Beispiel Einfahrten mit Türen ein Problem für das hier vorgestellte Erkennungsverfahren darstellen. Es ist daher zu überlegen, ob Einfahrten zusätzlich noch mit Türen gekennzeichnet werden sollten. Diese Kennzeichnung könnte dazu führen, dass auch mit geschlossenen Türen umgegangen werden kann. Türen könnten dabei in verschiedenen Zuständen an der Einfahrt angebracht sein. Diese Zustände wären dann zum Beispiel „offen“, „geschlossen“ oder etwas dazwischen.

Bei der hier verwendeten Pfadplanung wurde die kürzeste Strecke gewählt. Es könnte sich aber als Sinnvoll erweisen, einige Modelle anderen vorzuziehen, da die Erkennung auf den Sensordaten des Roboters mit für diese Modelle besser funktioniert.

Literaturverzeichnis

[1] C. Freksa. Using orientation information for qualitative spatial reasoning. In A. U. Frank, I. Campari, and U. Formentini, editors, *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, LNCS 639*, pages 162–178. Springer-Verlag: Berlin, 1992.

[2] C. Freksa and K. Zimmermann. On the utilization of spatial structures for cognitively plausible and efficient reasoning. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Chicago, 18-21 October 1992.

[3] B. J. Kuipers and Y.-T. Byun. A robust qualitative method for robot spatial learning. pages 774–779, 1988.

[4] P.-E. Michon and M. Denis. When and why are visual landmakrs used in giving directions? In D. R. Montello, editor, *Spatial Information Theory. Lecture Notes in Computer Science 2005*, pages 292–305, Berlin, 2001. Springer-Verlag.

[5] M. Raubal and S. Winter. Enriching wayfinding instructions with local landmarks. In M. Egenhofer and D. Mark, editors, *Geographic Information Science - Second International Conference GIScience 2002*, pages 243–259, Berlin, 2002. Springer.

[6] M. Wertheimer. Über Gestalttheorie. Vortrag vor der KANT-Gesellschaft, Berlin, am 17. Dezember 1924. In *Philosophische Zeitrschrift für Forschung und Aussprache, 1*, pages 39–60, 1924.

[7] D. Wolter, L. J. L. und Rolf Lakämper, and X. Sun. Shape-based robot mapping. In *Proceedings of the 27th German conference on artificial intelligence (KI-2004)*, 2004.

[8] L. A. Zadeh. From computing with numbers to computing with words. from manipulating of measurements to manipulation of perceptions. In *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions, Vol. 46, Issue 1*, pages 105–119, 1999.

Abbildungsverzeichnis

Anhang A

EBNF der Wegbeschreibungssprache

```
Wegplan          ::= {Schritt}.

Schritt          ::= Wandschritt | Einfahrtsschritt | Drehung |
                     Korridorschritt | Kreuzungsschritt.

Wandschritt      ::= "Fahre " Seite " entlang der Wand.".

Seite            ::= "rechts" | "links".

Einfahrtsschritt ::= Hineinfahren | Vorbeifahren.

Hineinfahren     ::= "Fahre durch die Einfahrt.".

Vorbeifahren     ::= "Fahre an der Einfahrt vorbei.".

Zifferohne0      ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

Ziffer           ::= "0" | Zifferohne0.

Drehung          ::= "Drehe dich " ("nach " Seite | "um") ".".

Korridorschritt  ::= "Fahre bis " Ziel ".".

Ziel             ::= "zur " Zifferohne0 {Ziffer}
                     (". Kreuzung " | ". Einfahrt " Seite) |
                     " zum Ende des Korridors".

Kreuzungsschritt ::= "Fahre " ("nach " Seite | "geradeaus") ".".
```

Anhang B

Karte der 5. Ebene

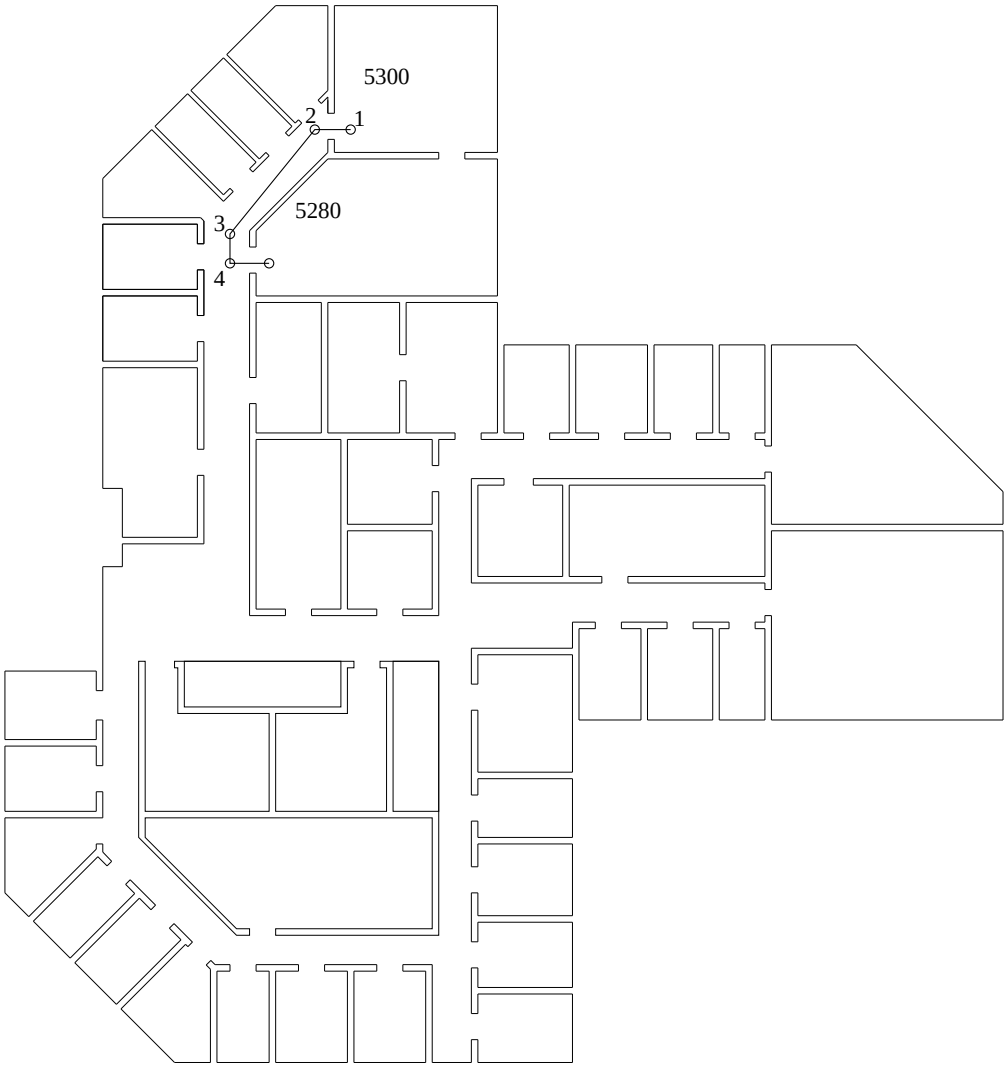

Abbildung B.1: Karte der 5. Ebene

Anhang C

Testkarte

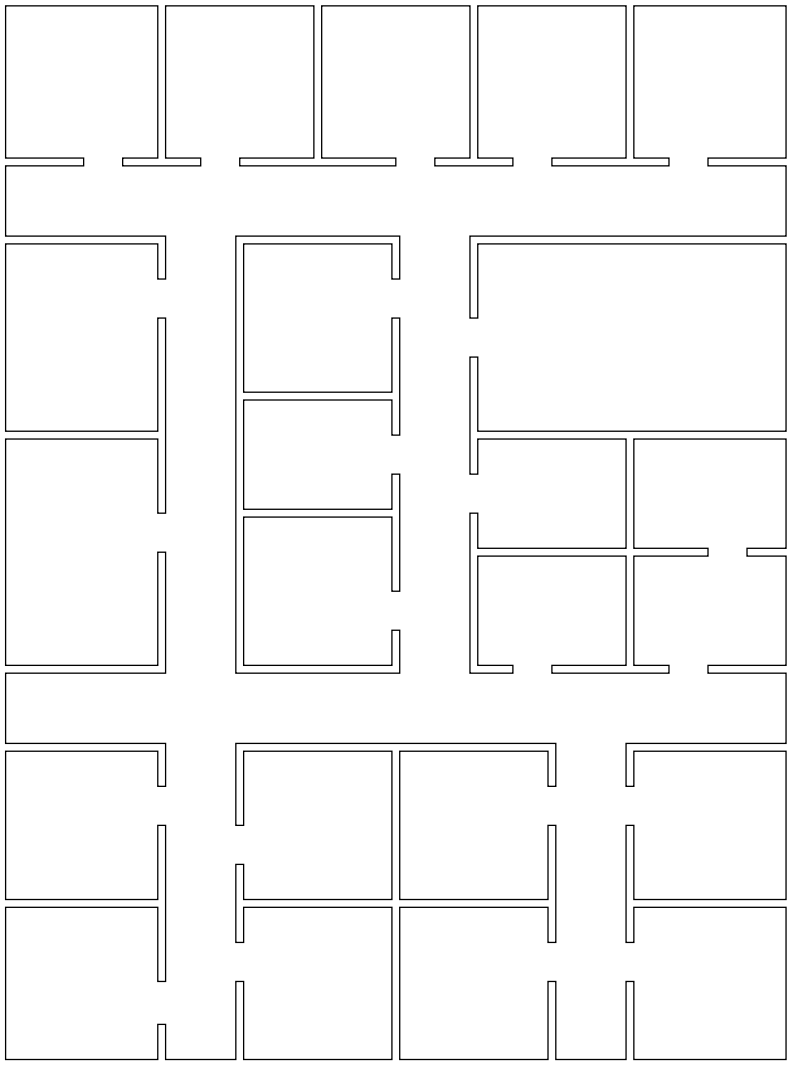

Abbildung C.1: Die in den Abschnitten 5.4 und 6.1 benutzte Testkarte

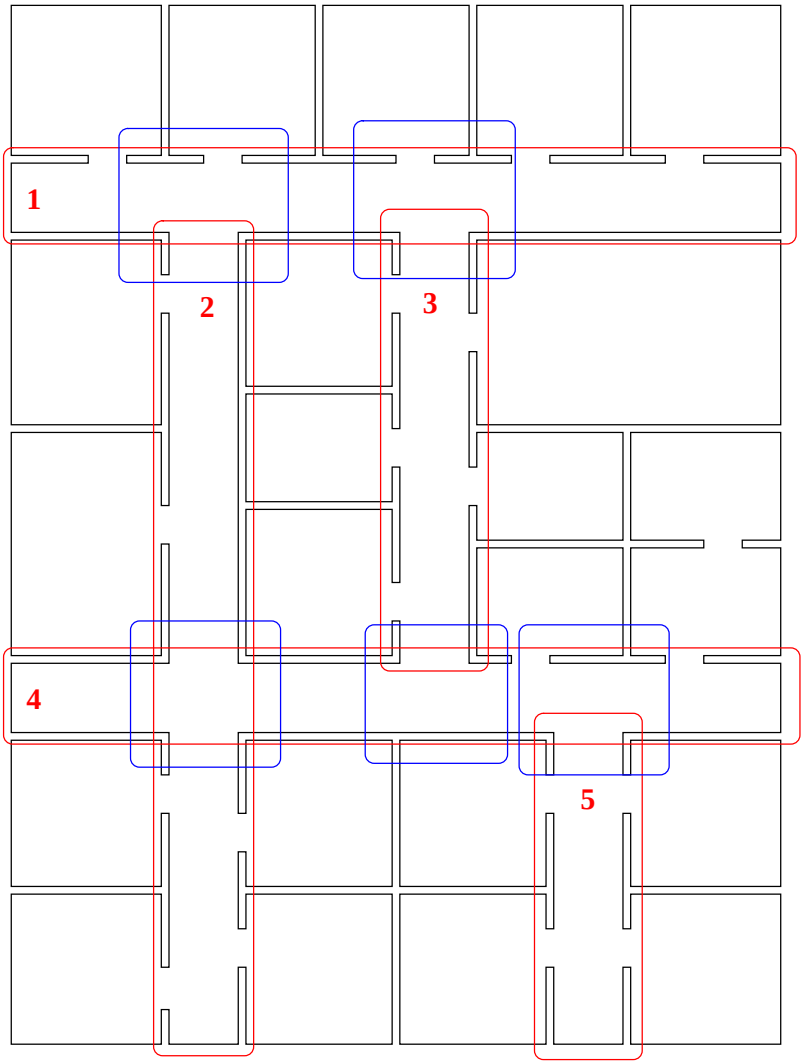

Abbildung C.2: Testkarte mit den zu erkennenden langen Korridoren und Kreuzungen

Anhang D

Der komplett erzeugte Testgraph

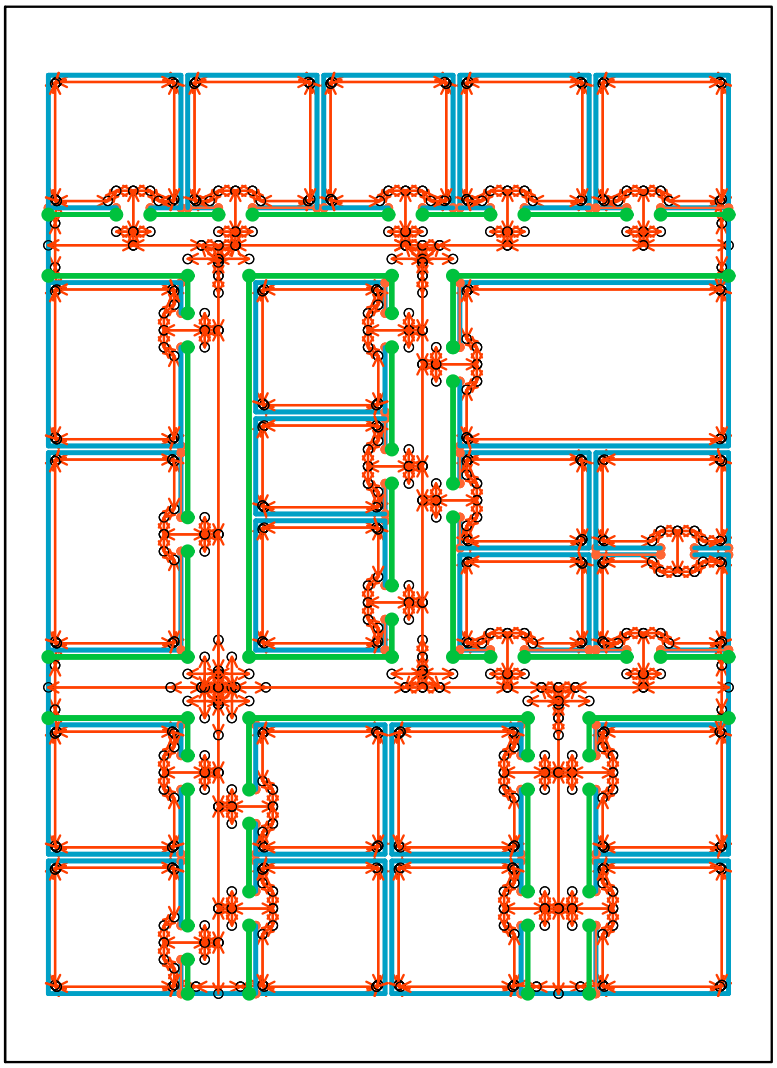

Abbildung D.1: Der in Abschnitt 6.1 erzeugte Graph