RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT BONN
INSTITUT FÜR INFORMATIK I

Bernd Brüggemann

Entkommen aus unbekannten Labyrinthen mit Einbahnstraßen

1. November 2006

Diplomarbeit

1. Gutachter: Prof. Dr. Rolf Klein
2. Gutachter: Prof. Dr. Norbert Blum

Erklärung

Mit der Abgabe der Diplomarbeit versichere ich gemäß §19 Absatz 7 der DPO vom 15. August 1998, dass ich die Arbeit selbstständig durchgeführt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Bonn, den 1. November 2006 Bernd Brüggemann

Top Red Guard: *Well the only way out of here*
is to try one of these doors!
Top Blue Guard: *One of them leads to the castle*
at the end of the labyrinth,
and the other one leads to... .
Top Red Guard: *bom bom BOM BOM*
Top Blue Guard: *Certain DEATH!*
all 4 guards at once: *Ooooooooooooohhhhh!*

aus: Reise ins Labyrinth, 1986, Regie: Jim Henson

Danksagung

An dieser Stelle möchte ich den vielen Leuten danken, die es ermöglicht haben, dass ich diese Diplomarbeit schreiben konnte, oder mehr noch, dass ich mein Studium erfolgreich beenden kann. Mein Problem dabei ist, dass es so viele Menschen gibt, denen ich Danken möchte und denen ich Dank schulde, als dass ich sie alle hier aufführen könnte. Und so werden auch in dieser Danksagung nur wenige namentlich genannt werden können. An alle Freunde und Weggefährten, die mich in der Universität begleitet haben, dies ist mein Dank an Euch.

Zu erst möchte ich Professor Klein danken, dass er es mir ermöglicht hat, in seiner Abteilung diese Arbeit zu schreiben. Bei Doktor Elmar Langetepe möchte ich mich für dieses interessante Thema bedanken. Durch ihn habe ich eine Diplomarbeit anfertigen können, die einerseits in der Zielrichtung der Abteilung liegt, anderseits auch meine Interessen berücksichtigt. Für die Zeit und die sehr gute Betreuung während meiner Diplomarbeit danke ich Doktor Tom Kamphans. Er hat viel Geduld und eine gutes Zuhörvermögen beweisen. Dies hat mir sehr geholfen.

Im Besonderen möchte ich allerdings zwei Personen danken. Dies ist zum einen Claudia Römer: Sie hat nicht nur die Auf und Abs meiner Stimmung während der Diplomarbeit ertragen, auf ihre Unterstützung darf ich mich schon seit Beginn meines Studiums verlassen. Sie gab mir den Ausgleich, wenn das Studium nicht so gut lief und motivierte mich, wenn meine Anstrengungen nachzulassen drohten. Die zweite Frau der ich noch viel mehr, als nur mein Studium zu verdanken habe, ist meine Mutter Angelika Brüggemann. Für ihre grenzenlose Unterstützung und den festen Glauben in mich und meine Fähigkeiten, auch wenn ich daran zweifelte, möchte ich danken.

Ich bedanken mich bei den vielen Leuten, die ich mit meine Ideen und meine Beweisvorschlägen nerven durfte und bei denen, die ich auf Rechtschreibfehler - Jagd schicken durfte. Auch hier kann ich nur einige Namen stellvertretend nennen; die nicht dabei sind, seid mir nicht böse, ich hab euch nicht vergessen. Ich danke: Martin Köhler, Björn Krüger, und Bernd Schönbach.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Das Navigieren durch Labyrinthe ist seit dem 18. Jahrhundert ein Problem, welches erst die Mathematik und später die Informatik beschäftigte. Schon das Problem der Königsberger Brücken kann als Wegfindungsproblem betrachtet werden. Bei der Navigation gibt es verschiedene Zielsetzungen. Ebenso kann die Ausgangslage unterschiedlich sein: Ist das Labyrinth bekannt, kann ein kürzester Weg von A nach B gesucht werden oder es es kann nach einem Weg gesucht werden, der es dem Roboter ermöglicht die gesamte innere Fläche des Labyrinths zu „sehen“.
Ebenso interessant ist es, Probleme zu betrachten, bei denen der Roboter vorher keine Information über das Labyrinth besitzt. Das Labyrinth ist also unbekannt. Wenn das Labyrinth in Zellen unterteilt ist, kann nach „Käse“ gesucht werden (RKSI93) oder es wird ein möglichst optimaler Weg gesucht, der jede Zelle mindestens einmal besucht (Kam05). Wenn das Labyrinth nicht diskretisiert ist, ergeben sich weitere interessante Probleme. So lösen einige Algorithmen beispielsweise das Problem, wie ein Roboter einen Punkt in einem Labyrinth findet, unter der Voraussetzung, der Roboter kennt nur die Richtung und die Entfernung zu seinem Ziel (LS86), (SV90a), (SV90b), (SV90c). Und natürlich die typische Frage wenn er sich in einem unbekannten Labyrinth befindet:

Wie findet der Roboter den Ausgang?

Mit der Annahme, dass keine Hilfsmittel wie z.B. eine Karte oder Markierungen im Labyrinth erlaubt sind, wird diese Aufgabe vom *Pledge-Algorithmus* für einfache Labyrinthe zuverlässig gelöst. Der Pledge-Algorithmus findet beweisbar immer den Ausgang (Ad80). Wenn man allerdings versuchen würde, mit Hilfe des Pledge-Algorithmus z.B. die Innenstadt mit dem Auto zu verlassen, so stößt man auf Grenzen des Pledge-Algorithmus. *Einbahnstraßen* können dazu führen, dass der Pledge-Algorithmus nicht mehr zum Ziel führt.
Diese Diplomarbeit stellt zwei Algorithmen vor, die den Pledge-Algorithmus so erweitern, dass ein Roboter es schafft, ein Labyrinth mit Einbahnstraßen zu verlas-

sen. Zusätzlich wird eine praktische Erprobung eines dieser Algorithmen auf dem Khepera II Roboter vorgestellt.
Kapitel 2 zeigt Grundlagen der Bewegungsplanung für Roboter und Grundlagen der Robotik auf. Hier werden bekannte Algorithmen vorgestellt, wobei ein Schwerpunkt auf den Pledge-Algorithmus gelegt wird. Zusätzlich werden noch Verfahren der Positionsbestimmung eines Roboters vorgestellt. Hier liegt der Schwerpunkt auf der Odometrie. Zuletzt wird der Khepera II vorgestellt und seine Handhabung beschrieben.
Kapitel 3 erweitert Labyrinthe um das Konstrukt der Einbahnstraße. Es werden Strukturen und Eigenschaften solcher Labyrinthe untersucht und definiert. Danach wird gezeigt, dass der bekannte Pledge-Algorithmus durch Modifikationen zu einem Einbahnstraßen-Pledge erweitert werden kann. Diese führen zu zwei Algorithmen, die beweisbar den Ausgang aus einem Labyrinth mit Einbahnstraßen finden. Einer dieser Algorithmen kommt mit dem Robotermodell des Pledge-Algorithmus aus, ist allerdings, durch seine hohe Laufzeit, für einen praktischen Einsatz am Roboter eher ungeeignet. Der andere Algorithmus benötigt ein erweitertes Robotermodell. Der Roboter benötigt die Fähigkeit, Einbahnstraßen voneinander zu unterscheiden. Mit Hilfe dieser zusätzlichen Information wird der Algorithmus vorgestellt, der auch praktisch einsetzbar ist.
Kapitel 4 dokumentiert die Implementierung des Pledge Algorithmus und des in Kapitel 3 entwickelten Algorithmus auf der Khepera II Plattform.
Kapitel 5 fasst die Ergebnisse dieser Arbeit zusammen. Hier wird betrachtet, inwieweit auf diese Ergebnisse aufgebaut werden kann und welche Fragen sich aus den vorgestellten Ergebnissen ergeben können.

Abbildung 1.1: *Einbahnstraße*

Kapitel 2

Grundlagen

2.1 Algorithmik

2.1.1 Labyrinthe

Was ist ein Labyrinth?

Das Wort „Labyrinth“ stammt ursprünglich aus der griechischen Mythologie. Dort bezeichnete „da-pu-ri-to“ (ausgesprochen etwa „laburintos“) den Palast des Minos, ein von Daidalos errichtetes Gebäude. Dieses Gebäude soll so ineinander verschlungene Gänge gehabt haben, dass niemand je wieder heraus fand. Letztendlich hat, der griechischen Sage nach, allerdings auch der Minotaurus (eine Hybride zwischen Mensch und Stier) dazu beigetragen, dass das Labyrinth des Minos unlösbar blieb.
Erst Theseus vermochte sowohl den Minotaurus zu überwinden, als auch mit Hilfe des Ariadnefadens, dem Labyrinth zu entkommen.
Labyrinthe faszinieren den Menschen seit jeher. Im Mittelalter wurden in Kirchen häufig Labyrinthe als Mosaike in Fußböden eingelassen. Ein berühmtes Fingerlabyrinth befindet sich am Eingang des Domes zu Lucca (Norditalien).
Im Barock wurden Irrgärten erstmals bekannt. Sie prägten das, was heutzutage von den meisten Menschen als Labyrinth aufgefasst wird. Im Gegensatz zu den ursprünglichen Labyrinthen bestehen Irrgärten aus vielen Abzweigungen und Sackgassen, so dass die Gefahr besteht sich zu verirren. Prägend sind hier vor allem die groß angelegten Gartenlabyrinthe der Aristokratie in der Renaissance, die damals der Zerstreuung und der Unterhaltung dienten. Diese engen Gänge, meist rechtwinklig angelegt, sind das Bild, das die meisten Menschen vor sich haben, wenn sie an ein Labyrinth denken.
Für diese Diplomarbeit wird allerdings eine wesentlich weiterfassende Definition verwendet (siehe (Kle05)):

Definition 2.1*: Wände*
Wände sind eine endliche Menge polygonaler Ketten $P_0, \ldots, P_n$. Je zwei dieser Ketten dürfen sich nicht schneiden. Diese Ketten bilden die Ränder der Hindernisse.

Die durch Wände erzeugten *Hindernisse* müssen nicht massiv sein.

Definition 2.2*: Innenhöfe*
Innenhöfe sind freie Flächen, die komplett von Wänden eingeschlossen sind (vgl. Abb 2.1). Innenhöfe können selbst wieder Hindernisse enthalten, welche neue Innenhöfe bilden.

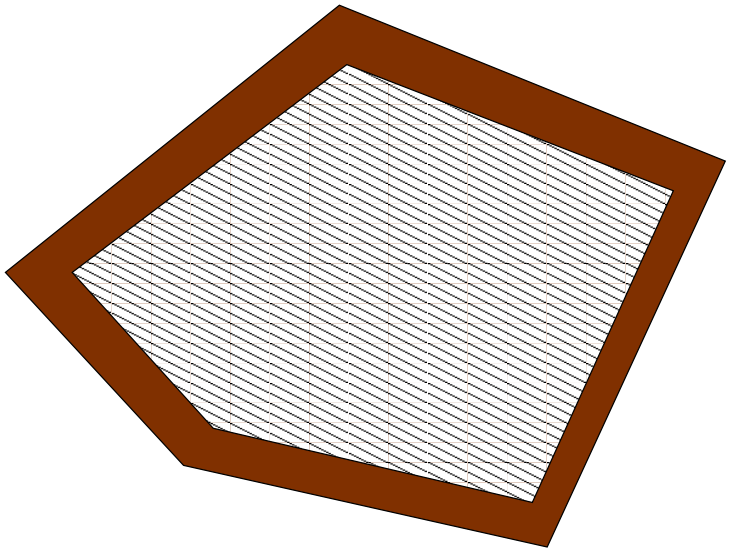

Abbildung 2.1: *Ein Beispiel für einen Innenhof. Die schraffierte Fläche ist keine Wand, folglich freie Fläche. Da sie aber vollständig von Wänden eingeschlossen ist, ist sie ein Innenhof.*

Definition 2.3*: Labyrinth*
Ein Labyrinth ist eine Unterteilung der Ebene in Hindernisse und freie Fläche.

Definition 2.4*: C_{frei}*
C_{frei} ist die unbeschränkte freie Fläche des Labyrinthes.

In den hier betrachteten Labyrinthen wird der Roboter an einer bestimmten Stelle gestartet und muss nun den Ausgang finden.
Nach der Definition eines Labyrinthes existiert eine unbeschränkte Zusammenhangskomponente. Diese Zusammenhangskomponente ist freie Fläche. In ihr sind Wände (Hindernisse) eingebettet. Nur wenn der Roboter in diesem unbeschränkten Gebiet startet, kann er das Labyrinth verlassen.

Definition 2.5*: Ausgang aus einem Labyrinth*
Der Ausgang eines Labyrinthes ist die Linie, die der Roboter überfahren muss, um das Signal „entkommen“ zu bekommen. Dies ist normalerweise dann der Fall, wenn der Roboter das umschließende Rechteck (die „Bounding Box“) der Szenerie überschreitet.

Ob sich ein Roboter innerhalb des Labyrinthes oder außerhalb befindet, autonom von diesem Roboter überprüfen zu lassen ist schwierig (besonders bei den beschränkten Sensorfähigkeiten eines realen Roboters). Deshalb wird angenommen, der Roboter bekommt von außen ein Signal, wenn er das Labyrinthinnere verlassen hat.

2.1.2 Der Pledge-Algorithmus

Der Pledge-Algorithmus ist ein wohlbekannter Algorithmus, um aus einem Labyrinth zu entkommen. Dem Buch „Turtle Geometrie“ zufolge hat diesen Algorithmus ein 12 jähriger Junge (John Pledge) erfunden (siehe (Ad80)). Da dieser Algorithmus essentielle Bedeutung für diese Diplomarbeit hat, werden im Folgenden sowohl der Algorithmus, als auch die Beweise seiner Korrektheit aufgeführt, so wie sie z. B. in (Kle05) zu finden sind.

Die Idee

Der zur Verfügung stehende Roboter hat nur rudimentäre Funktionen. Diese sind:

- Vorwärts fahren
- drehen
- den Drehwinkel mitzählen
- ein Hindernis direkt vor sich erfassen
- sich an diesem Hindernis entlangtasten

In (Kle05) wird dies treffend in Bezug auf die Mythologie des Labyrinthes, wie folgt beschrieben:
Das bedeutet für den antiken Helden: Er wacht im Inneren des Labyrinthes auf und verfügt über keinerlei Hilfsmittel wie Ariadnefäden oder ein Werkzeug zum Anbringen von Markierungen. Außerdem ist es so dunkel, dass er sich alleine auf seinen Tastsinn verlassen muss.

Algorithm 1 Pledge-Algorithmus

```
 1: wähle Richtung beliebig
 2: Winkelzähler := 0;
 3: repeat
 4:     repeat
 5:         folge Richtung
 6:     until Wandkontakt;
 7:     repeat
 8:         folge der Wand, mit der Wand auf der linken Seite
 9:     until Winkelzähler = 0;
10: until Entkommen
```

Die Idee des Pledge-Algorithmus ist die, dass der Roboter sich so lange geradeaus bewegt, bis er auf ein Hindernis stößt. Sobald ein Hindernis auftaucht, geht der Pledge-Algorithmus in einen „Wall-Follower Modus“. Nach Vereinbarung hält der Roboter die Wand des Hindernisses immer links von sich. Jede Drehung wird mitprotokolliert (z. B. durch Odometrie oder einen Kompass). Erreicht dieser Winkelzähler wieder 0, so löst sich der Roboter vom Hindernis und fährt wieder vorwärts. Dieses Vorgehen findet immer den Ausgang.

Beweis der Korrektheit des Pledge-Algorithmus (Kle05)

Theorem 2.6*: Der Pledge-Algorithmus findet in jedem Labyrinth von jeder Startposition aus einen Weg ins Freie, wenn überhaupt ein Ausweg existiert.*

Sei L ein Labyrinth und s eine Startposition in L. Zunächst wird gezeigt, dass der Winkelzähler des Roboters nur Werte eines bestimmten Bereiches annehmen kann.

Lemma 2.7*: Der Winkelzähler nimmt niemals einen positiven Wert an.*

Beweis. Der Winkelzähler wird beim Start des Algorithmus auf Null gesetzt und sein Wert ist auch später gleich Null, so lange der Roboter sich nicht im Wall-Following Modus befindet.

Wenn der Roboter nun auf ein Hindernis trifft, führt er zunächst eine Rechtsdrehung aus, um der Wand folgen zu können. Hierdurch erhält der Zähler einen negativen Wert. Sobald der Zähler wieder die Null erreicht, löst sich der Roboter vom Hindernis und fährt geradeaus weiter. Aus Stetigkeitsgründen kann der Winkelzähler niemals einen positiven Wert erreichen. ■

Um zu zeigen, dass der Pledge-Algorithmus immer einen Weg zum Ausgang findet, wenn es einen gibt, wird angenommen, dass der Roboter von seinem Startpunkt s aus keinen Weg ins Freie findet. Nun soll gezeigt werden, dass in diesem Fall kein Ausgang existieren kann.
Wenn der Roboter keinen Ausgang findet, terminiert der Pledge-Algorithmus nicht. Deswegen legt der Roboter in diesem Fall einen unendlich langen Weg zurück. Die Struktur dieses Weges wird näher betrachtet:

Lemma 2.8*: Angenommen, der Roboter findet nicht aus dem Labyrinth heraus. Dann besteht sein Weg, bis auf ein endliches Anfangsstück, aus einem geschlossenen Weg, der immer wieder durchlaufen wird.*

Beweis. Der Weg des Roboters ist eine polygonale Kette, deren Eckpunkte aus einer endlichen Menge stammen:

- Die Eckpunkte von Wänden im Labyrinth
- Von jeder Wandecke aus, in Startrichtung des Roboters, der erste Punkt auf dem nächsten Hindernis

Anders ausgedrückt, nur an den Ecken eines Hindernisses kann der Roboter seine Bewegungsrichtung ändern. Auf der freien Fläche gibt es keine Richtungsänderung, da der Algorithmus hier nur eine Gerade-aus-Fahrt vorsieht. Ebenso kann keine Richtungsänderung erfolgen, wenn der Roboter an einer Hinderniskante entlang fährt. Eine Richtungsänderung an einer Hinderniskante wird durch das Verhalten des Wall-Followers ausgeschlossen.
Falls der Roboter einen solchen Eckpunkt ein zweites Mal mit demselben Winkelzählerstand besucht, wiederholt er den Weg dazwischen zyklisch immer wieder, da sein Verhalten deterministisch ist, und es folgt die Behauptung des Lemmas.
Wenn aber jeder Eckpunkt höchstens einmal mit demselben Winkelzählerstand besucht wird, erreicht der Roboter nur endlich oft Eckpunkte mit dem Winkelzählerstand Null. Sobald diese Besuche erfolgt sind, kann der Roboter nie wieder eine freie Bewegung ausführen; er folgt danach also nur noch einer einzigen Wand, und sein Weg ist auch in diesem Fall zyklisch.

■

Nun wird der geschlossene Weg P, den der Roboter bei seinem vergeblichen Versuch aus dem Labyrinth zu entkommen, laut Lemma immer wieder durchlaufen. Es muss also gezeigt werden, dass es keinen Ausweg gibt.

Lemma 2.9*: Der Weg P kann sich nicht selbst kreuzen.*

Beweis. Wenn P sich selbst kreuzen würde, müsste es ein Segment B von P geben, das in einem Punkt z auf ein anderes Segment A auftrifft. Eines der beiden Segmente, z. B. B, muss frei sein, da sich die Wände der Hindernisse nicht schneiden können (vgl. Abb. 2.2).
Seien $W_A(\acute{z})$ und $W_B(\acute{z})$ die Winkelzählerstände an einem Punkt $\acute{z}$ kurz hinter dem Punkt z bei Anreise über A bzw. B. Dann ist

$$W_B(\acute{z}) = -\beta \text{ mit } 0 \leq \beta < \pi$$
$$W_A(\acute{z}) = -\beta + 2k\pi \text{ mit } k \in Z$$

Dies gilt, da hinter z die Ausrichtung des Roboters stets dieselbe ist. Wäre $k \geq 1$, so ergäbe sich

$$W_A(\acute{z}) = -\beta + k2\pi > -\pi + 2\pi = \pi$$

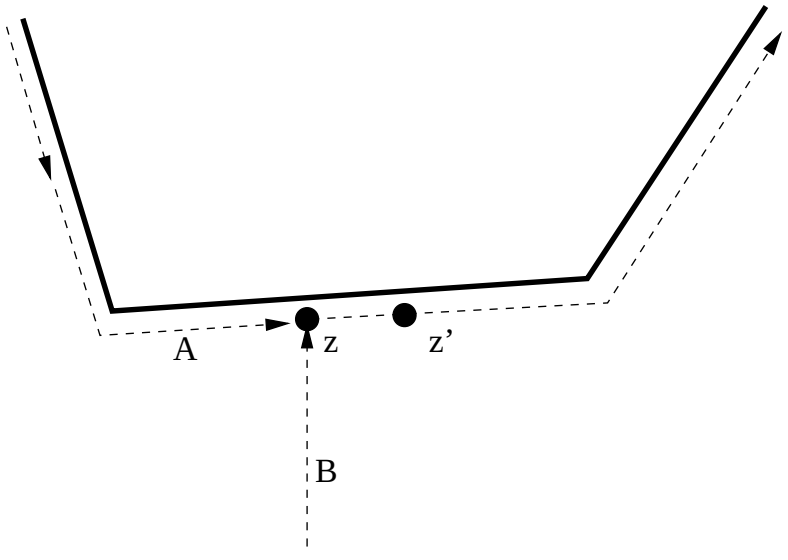

Abbildung 2.2: *Segment B trifft auf Segment A*

Dies ist ein Widerspruch, da der Winkelzähler niemals positive Werte annehmen kann. Also muss $k \leq 0$ sein.
Aus $k = 0$ würde $W_A(\acute{z}) = W_B(\acute{z})$ folgen. In diesem Fall würden sich die Wegstücke über A und B hinter z niemals wieder trennen. Wenn also nach $\acute{z}$ als Nächstes etwa das Segment B besucht würde, käme das Segment A niemals wieder an die Reihe. Dies steht allerdings im Widerspruch dazu, dass sowohl A wie auch B Teile des Weges P sind, und P unendlich oft durchlaufen wird.

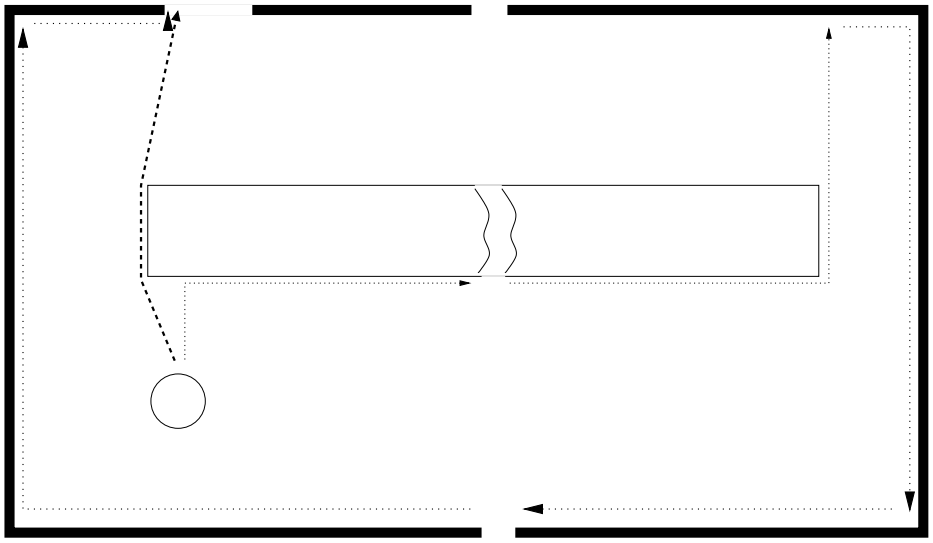

Abbildung 2.3: *Die gepunktete Linie ist der Weg, den der Roboter mit Hilfe des Pledge-Algorithmus geht. Die gestrichelte Linie ist der kürzeste Weg. Da das Hindernis beliebig lang werden kann, ist der Quotient aus der Länge des Weges, des Pledge-Algorithmus, und dem optimalen Weg beliebig groß. Dies bedeutet, das es keinen konstanten kompetitiver Faktor für den Pledge-Algorithmus gibt.*

Also bleibt nur der Fall $k < 0$ übrig. Dann ist $W_A(t) < W_B(t)$ für alle Punkte t von $\acute{z}$ bis zu dem Punkt v, an dem sich die Wege wieder trennen; dort muss dann $W_B(v) = 0$ sein. Es existiert also keine echte Kreuzung, sondern nur eine Berührung. Die Wegstücke A und B liegen eine Zeit lang nebeneinander.

■

Um nun zu zeigen, dass der Pledge-Algorithmus immer einen Ausgang findet, werden zwei Fälle betrachtet: Angenommen, der Roboter durchliefe den Weg P *gegen* den Uhrzeigersinn. Dann wird sich der Winkelzähler bei jedem Durchlauf um 2π *erhöhen*. Während jeden Durchlaufs führt der Zählerstand dieselben Auf- und Abwärtsschwankungen aus; wenn der Wert aber insgesamt bei jeder Runde um 2π zunehme, müsste er irgendwann positiv werden, was jedoch, wie gezeigt, nicht möglich ist.
Also durchläuft der Roboter den Weg P *im* Uhrzeigersinn und der Wert des Winkelzählers nimmt bei jedem Umlauf um 2π ab. Dann können irgendwann nur noch echt negative Werte angenommen werden. Deshalb kann der Weg P keine freien Segmente, also Teilstücke, die nicht an einer Hinderniswand entlang führen, enthalten. Wegen der Umlaufrichtung im Uhrzeigersinn, kann es sich dabei nur um die Wand eines Innenhofs handeln, in dem der Roboter gefangen ist.

Beweis. zu Theorem 2.6
Damit ist gezeigt, dass der Pledge-Algorithmus immer einen Weg aus einem Labyrinth findet, wenn es diesen Weg gibt. ■

Beispiele für die Verhaltensweise des Pledge-Algorithmus

Der Pledge-Algorithmus findet, wie vorher gezeigt, immer einen Ausweg, so es denn einen gibt. Für den Weg, den der Pledge-Algorithmus zurücklegt, gibt es keinen konstanten kompetitiven Faktor. Das bedeutet, das Verhältnis der Länge des Weges, den der Pledge-Algorithmus nimmt, zur optimalen Länge ($\frac{W_{Pledge}}{W_{opt}}$), kann beliebig groß werden (vgl. Abb 2.3). Da dem Pledge-Algorithmus nur ein „Tastsinn“ und der Winkelzähler zu Verfügung stehen, werden in manchen Labyrinthen recht aufwendige Wege beschritten. Ein sehr gutes Beispiel hierfür ist ein spiralförmiges Hindernis (vgl. Abb. 2.4). Der Roboter startet in der Mitte. Bei der Wandverfolgung dreht sich der Winkelzähler immer weiter auf, so dass, beim Erreichen des Spiralen-Außenrandes der Roboter wieder in die Spirale einfährt. Allerdings nun nicht mehr bis ganz ins Innere. So dreht sich der Roboter langsam aus der Spirale heraus, bis der Winkelzähler einen Wert erreicht hat, der ihm ermöglicht, sich an der Außenwand der Spirale zu lösen.

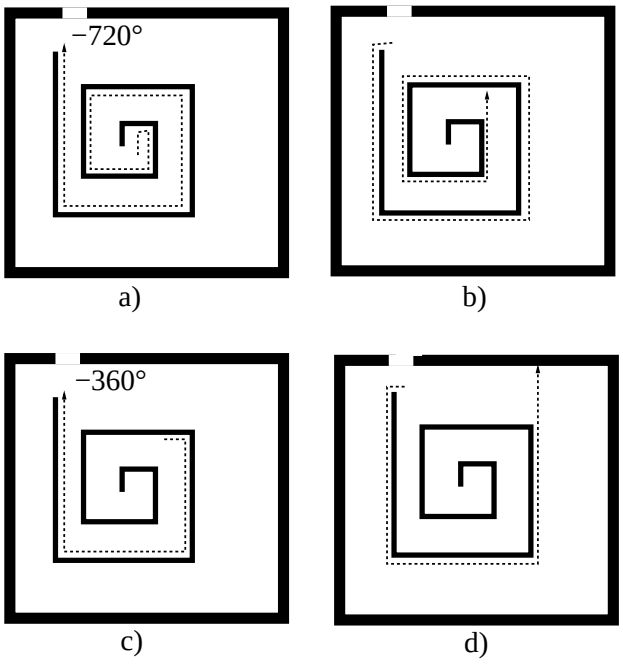

Abbildung 2.4: *Der Weg des Pledge-Algorithmus durch eine Spirale. Erst durch wiederholtes Einfahren in das Spiraleninnere, wird der Winkelzähler groß genug, damit der Roboter sich von der Außenwand der Spirale löst.*

2.1.3 Weitere Strategien um in einem Labyrinth zu navigieren

Shannons Maus

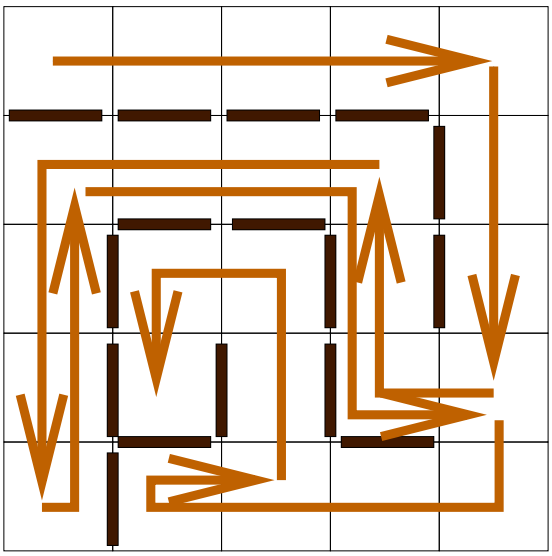

Abbildung 2.5: *Der Weg von Shannons Maus durch ein Beispiellabyrinth. Es besteht aus fünf mal fünf Feldern. Felder können durch Wände voneinander getrennt werden.*

In (RKSI93) wird Shannons Maus als eine Möglichkeit beschrieben, in einem fünf mal fünf Quadrate großem Feld ein Stück Käse zu finden. Der Algorithmus (zuerst beschrieben in (Sha52)) fährt alle Quadrate ab, um jenes Quadrat zu finden, auf welchem der Käse liegt.
Das Labyrinth besteht aus Aluminiumwänden, die zwei benachbarte Quadrate voneinander trennen (vgl. Abb. 2.5). Für jedes Quadrat wird abgespeichert, in welche Richtung die Maus das letzte Mal das Quadrat verlassen hat. Wenn nun die Maus auf ein Quadrat kommt, versucht sie das Quadrat $90°$ gegen den Uhrzeigersinn zu der gespeicherten Richtung wieder zu verlassen. Das bedeutet: Ist die Maus beim letzten Mal nach Norden aus dem Feld gelaufen, so will sie nun das Quadrat nach Westen hin verlassen. Richtungen die durch Wände versperrt sind, werden nicht berücksichtigt. Für Felder, die das erste Mal besucht werden, wird eine Vorzugsrichtung angegeben, z. B. dass ein Quadrat, welches zum ersten Mal besucht wird, nach Norden verlassen werden soll.
Shannons Maus läuft alle 25 Felder des Labyrinthes ab. Wenn nun die Markierung kei-

nen Käse darstellt, sondern einen Ausgang, findet die Maus also auch den Ausgang. Interessant an diesem Ansatz ist außerdem, dass Shannon diese Maus als Automaten gebaut hat. Das Modell der Maus war elektromechanisch und die Steuerung wurde allein über Relais realisiert.

Lumelskys Algorithmen

In (LS86) wurden die beiden Algorithmen *Bug1* und *Bug2* vorgestellt. Die Problemstellung ist die, dass der Roboter durch ein Arrangement von Hindernissen einen Zielpunkt erreichen soll. *Bug1* ist wie folgt beschrieben:
Der Roboter trifft das i-te Hindernis am Auftreffpunkt H_i und verlässt es am Punkt (*leave point*) L_i, $i = 1, 2, \ldots$. L_0 ist der Startpunkt.

1. Der Roboter bewegt sich von Punkt L_{i-1} in gerader Linie auf das Ziel zu. Entweder erreicht der Roboter das Ziel, dann terminiert der Algorithmus. Oder der Roboter trifft auf ein Hindernis. Dann ist dieser Auftreffpunkt H_i und der Algorithmus fährt mit Punkt 2 fort.

2. Der Roboter fährt am Hindernis entlang, so dass das Hindernis immer auf der rechten Seite des Roboters liegt. Entweder der Roboter erreicht das Ziel, dann terminiert der Algorithmus. Oder das Hindernis wird einmal umlaufen, d.h. der Roboter erreicht wieder H_i. Dann wird L_i festgelegt. L_i ist der Punkt der Strecke, die um das Hindernis herumführt, die am nächsten zum Zielpunkt liegt.

3. Der Roboter fährt auf dem kürzesten Weg zu L_i. Dann wird i um eins erhöht und der Algorithmus macht mit Schritt 1 weiter.

Einen Beweis für die Korrektheit von Bug1 findet sich ebenfalls in (LS86).

Bug2 ist eine Vereinfachung des Bug1 Algorithmus. Er vermeidet die Mehrfachfahrten bei der Bestimmung des *leave point* L_i. Der Algorithmus startet in L_0, was dem Startpunkt entspricht:

1. Von L_{i-1} bewegt sich der Roboter auf der geraden Linie von S nach T (dem Zielpunkt). Wenn der Zielpunkt erreicht ist, terminiert das Programm. Ansonsten wird ein Hindernis gefunden. Der Auftreffpunkt des Hindernis wird wie in Bug1 als H_i definiert.

2. Der Roboter folgt wieder dem Hindernis so, dass das Hindernis auf der rechten Seite des Roboters liegt. Dies macht er so lange bis eines der folgenden Ereignisse eintritt:

- Der Roboter erreicht sein Ziel, der Algorithmus terminiert.
- Der Roboter erreicht wieder H_i. Dies bedeutet, dass der Roboter das Ziel nicht erreichen kann, da Roboter und Zielpunkt nicht in der gleichen Zusammenhangskomponente des Labyrinths liegen. Der Algorithmus bricht ab.
- Der Roboter erreicht den Punkt Q, der zum einen auf der Verbindungslinie ST liegt. Zusätzlich liegt Q näher an T als H_i an T. Die Linie QT schneidet das Hindernis nicht in Punkt Q. Erfüllt Q alle diese Eigenschaften wird es als L_i definiert und der Algorithmus macht mit Schritt 1 weiter. Die existenz eines Punktes, der alle diese Eigenschaften erfüllt, ist garantiert, wenn der Zielpunkt erreichbar ist.

Algorithm 2 Bug2

1: $T =$ *Zielpunkt*;
2: **repeat**
3: **repeat**
4: folge *Richtung T*
5: **until** *Hindernis*;
6: $L =$ *Auftreffpunkt*
7: **repeat**
8: folge der Wand;
9: $Q =$ *aktuelle Position*
10: **if** $Q == L$ **then**
11: Abbruch;
12: **end if**
13: **until** ($Q \in \overline{ST}$) $\wedge$ (*Distanz(Q,T)* $<$ *Distanz(Startpunkt,T)*) $\wedge$ ($\overline{QT} \cap$ *Hindernis* $= \emptyset$);
14: **until** *T erreicht*

Sankaranarayanans Algorithmus
Ebenso wie Bug1 und Bug2 stellten Sankaranarayanan und Vidyasagar verschiedene Algorithmen vor, die einen Zielpunkt in einem unbekannten Labyrinth finden (z. B. in (SV90a), (SV90b) und (SV90c)). Als Beispiel ist hier der Algorithmus *Alg2* angegeben werden (vgl. (SV90b) und (RKSI93):

1. Der Roboter bewegt sich auf gerader Linie auf den Zielpunkt T zu. Sobald er ein Hindernis berührt, wird das Hindernis so umfahren, dass das Hindernis immer rechts vom Roboter liegt. Der Berührungspunkt mit dem Hindernis wird H genannt.

2. Der Roboter verlässt ein Hindernis am Punkt L_i nur, wenn zwei Bedingungen erfüllt sind:

   - am Punkt L_i kann der Roboter auf gerader Linie in Richtung Zielpunkt T fahren, ohne gegen das Hindernis zu stoßen
   - L_i ist näher an T als alle Punkte, die der Roboter vorher schon besucht hat.

3. Wenn der Punkt H erreicht wird, wird von H aus ein noch nicht besuchter Bereich des Hindernis aufgesucht.

Bug1, Bug2 und Alg2 setzen voraus, dass der Roboter sowohl die Richtung, wie auch die Distanz zu seinem Zielpunkt kennt. Verglichen mit dem Pledge-Algorithmus sind sie also aufwendiger, haben allerdings auch die Fähigkeiten, bestimmte Punkte in einem Labyrinth zu finden. Der Pledge-Algorithmus kann ja „nur“ Ausgänge finden, d.h. einen Ausgang der am Rand des Labyrinths liegt. Falltüren die irgendwo im Inneren des Labyrinths einen Ausgang darstellen, kann der Pledge-Algorithmus nicht finden.
In (SV90c) unterteilen Sankaranarayanan und Vidyasagar die Algorithmen, die mit Hilfe von Berührungssensoren ihren Weg finden, in zwei Klassen. Algorithmen der Klasse 1 untersuchen erst das gesamte Hindernis, bevor sie sich wieder lösen. *Bug1* gehört in diese Klasse. Algorithmen der Klasse 2 fahren bei mindestens einem Hindernis nicht komplett um das Hindernis herum.
Die Algorithmen Bug1, Bug2, Alg2 sind *metrische* Algorithmen. Dies bedeutet, sie benutzen Informationen wie Position und Entfernung.
In (SM92) wird der Algorithmus Curve1 vorgestellt, welcher keine metrischen Informationen nutzt. Die Idee ist, einem gegebenen Pfad zu folgen, welcher durch eine Umgebung mit unbekannten Hindernissen verläuft:

- Sei ST eine Kurve die keine Selbstschnitte hat und Start- und Zielpunkt miteinander verbindet.

- ST kann nur eine gerade Anzahl von Schnittpunkten mit den unbekannten Hindernissen haben (Jordan-Curve Theorem).

- Sei C der Zähler für die Anzahl der Schnitte von ST mit den Hindernissen.

Der Algorithmus geht wie folgt vor:

1. Der Roboter startet am Punkt S. Setzte $C = 0$;
2. Der Roboter bewegt sich so lange an der Kurve entlang, bis eines der folgenden Ergebnisse eintritt:
   (a) Das Ziel T ist erreicht (Abbruch).
   (b) Ein Hindernis wird getroffen. Dann folgt der Roboter dem Hindernis so, dass die Wand immer auf der rechten Seite liegt.
3. Der Roboter folgt der Hinderniswand bis zu einem der folgenden Ereignisse:
   (a) Das Ziel T ist erreicht (Abbruch).
   (b) Die Kurve ST wird wieder getroffen. Dieser Trefferpunkt sei P. Dann wird je nach Zählerstand folgende Aktion ausgeführt:
      i. C ist gleich Null und der Roboter kann an P wieder der Kurve ST folgen. Dann folgt er der Kurve vom Hindernis weg.
      ii. C ist ungleich Null und der Roboter kann an P wieder der Kurve ST folgen. C wird um eins dekrementiert und der Roboter folgt weiter dem Hindernis. Der Algorithmus geht weiter mit Punkt 3.
      iii. An P kann der Roboter nicht der Kurve ST vom Hindernis weg folgen. C wird um eins inkrementiert und der Roboter folgt weiter dem Hindernis. Weiter mit 3.

Die hier vorgestellten Steuerungsalgorithmen sind nur eine kleine Auswahl der existierenden Algorithmen, um in Labyrinthen zu navigieren. So sind z. B. noch viele weitere Varianten des *Bug*- Algorithmus bekannt.

Universelles Steuerwort
Für die Theorie ist ebenfalls interessant, dass es ausreicht, wenn ein Roboter mit Tastsensoren und einem Zielkompass ausgerüstet ist, um einen Zielpunkt zu finden. Diese Idee wurde in (Hem93) von Hemmerling vorgestellt. Die hier gezeigte Zusammenfassung ist (Kle05) entnommen:

Theorem 2.10*: Im Prinzip genügen Zielkompass und Tastsensor, um in einer unbekannten Umgebung einen Zielpunkt zu finden.*

Beweis. Die Bewegung des Roboters können über drei Grundbefehle gesteuert werden:

T: Laufe Richtung Ziel, bis Ziel oder Hindernis erreicht

L: Folge der Wand, mit der Wand an der linken Seite, bis zur nächsten Ecke

R: Folge der Wand, mit der Wand an der rechten Seite, bis zur nächsten Ecke

Jeder Weg des Roboters, der mit diesen Steuerelementen beschrieben werden kann, lässt sich durch ein endliches *Steuerwort* über dem Alphabet $\Sigma = \{T, L, R\}$ darstellen.
Angenommen, der Zielpunkt t ist vom Startpunkt s aus erreichbar. Dies bedeutet, dass wenn die XY-Ebene längs der Wände aufgeschnitten wird, liegen s und t im Abschluss A des unbeschränkten Gebiets.
Wenn der Roboter in s startet, kann er nur zu anderen Punkten in A gelangen. Wird der Roboter durch ein Steuerwort über Σ gesteuert, sind es sogar nur endlich viele:

- Startpunkt s
- Hindernisecken
- Endpunkte von freie T-Bewegungen, die in s oder in Wandecken starten
- Zielpunkt t

Seien nun $\{p_1, p_2 \ldots, p_m\}$ die Menge dieser Punkte. Für jedes p_i gibt es ein Steuerwort $w(p_i)$, welches den Roboter von p_i ans Ziel bringt.
Die Beweisidee für Theorem 2.10 folgt aus dem Lemma:

Lemma 2.11*: Es gibt ein endliches universelles Steuerwort w über Σ, das den Roboter von jedem Punkt p_i aus zum Ziel führt.*

Beweis. Mit $w_1 = w(p_1)$ findet man von p_1 aus zum Ziel.
Versucht nun der Roboter dieses Wort von Punkt p_2 aus anzuwenden, kann es vorkommen, dass bestimmte Befehle nicht ausführbar sind. So kann z. B. die Steuerung „T“ dazu führen, dass der Roboter durch ein Hindernis fahren müsste. Solche Befehle werden ausgelassen. Es wird abgebrochen sobald der Zielpunkt erreicht ist.
Im Allgemeinen wird sich der Roboter nach einer solchen Anwendung von w_1 auf p_2 nicht im Zielpunkt befinden, sondern an irgendeinem anderen Punkt q_2. Nun kann er aber ans Ziel gelangen, wenn er seinen Weg mit $w(q_2)$ fortsetzt. Das Wort:

$$w_2 = w_1 \oplus w(q_2)$$

führt den Roboter also schon von zwei Punkten aus ans Ziel: sowohl von p_1 wie auch von p_2.
Dieses Verfahren kann induktiv fortgesetzt werden. Wenn dies für alle (endlich vielen) möglichen Punkte durchgeführt wird, erhält man ein Wort w_m, welches von allen möglichen Punkten aus zum Ziel führt.

■

Es existiert also ein universelles Steuerwort, welches den Roboter aus dem Labyrinth hinausbringt. Natürlich ist dieses Steuerwort dem Roboter unbekannt und ebenso natürlich ist dieses Steuerwort für jedes Labyrinth anders. Aber da es endlich ist, kann man auf der Suche nach diesem Steuerwort wie folgt vorgehen:
Der Roboter probiert alle Steuerworte über Σ der Länge eins aus. Wenn dies nicht ans Ziel geführt hat, versucht er alle Steuerworte über Σ der Länge zwei usw. Spätestens wenn der Roboter hierdurch das universelle Steuerwort gefunden hat, findet er auch das Ziel, da das universelle Steuerwort ihn ja von jedem möglichen Punkt aus ans Ziel t bringt. Es ist durchaus möglich, dass schon ein früheres Steuerwort durch Zufall den Roboter ans Ziel geführt hat, jedoch spätestens das universelle Steuerwort bringt ihn dahin. Da dieses Wort eine endliche Länge hat, findet der Roboter dieses auch in endlicher Zeit.

■

In (Hem93) wird auch die Laufzeit betrachtet. Hierzu muss man sich die endlich vielen möglichen Punkte anschauen, die der Roboter mit Hilfe eines Steuerwortes erreichen kann. Diese Punkte kann man sich als Knoten vorstellen. Die Knoten sind untereinander mit Kanten verbunden, wenn es einen Steuerbefehl gibt, der den Roboter von dem einen Knoten zu dem anderen bringt. Dies ist eindeutig, da der Roboter mit einem bestimmten Steuerbefehl (L, R *oder* T) von einem bestimmten Punkt im Labyrinth, zu einem bestimmten anderen Punkt kommt [1]. Es ergibt sich folgendes Lemma:

[1] Wenn ein Steuerbefehl nicht ausgeführt werden kann, bleibt der Roboter an seinem Standort

Lemma 2.12*: Die Länge des universellen Steuerwortes liegt in $O(n^2)$, wobei n die Anzahl der möglichen Positionen des Roboters ist.*

Beweis. Es gibt n verschiedene mögliche Positionen (Knoten) des Roboters.
Um von einem Knoten jeden beliebigen anderen Knoten (z. B. den Zielknoten) zu erreichen, muss der Roboter maximal n Steuerbefehle ausführen.
Da das universelle Steuerwort aus einer Hintereinanderschaltung von Steuerworten für jeden Knoten besteht, ist das universelle Steuerwort maximal n^2 groß.

■

Dies führt zu dem Lemma:

Lemma 2.13*: Es müssen maximal $3^{O(n^2)}$ Wörter untersucht werden, um zum Zielpunkt zu kommen.*

Beweis. Nach Lemma 2.11 existiert ein universelles Steuerwort und nach Lemma 2.12 hat es maximal die Länge n^2.
Dies bedeutet, dass der Roboter spätestens nachdem er alle Steuerworte der Länge n^2 ausprobiert hat, das Ziel erreicht. Die Anzahl der Wörter, die bis dahin ausprobiert wurden ist die Summe aller dreibuchstabigen Wörter der Länge eins bis zur Länge n^2:

$$\Sigma_{i=1}^{n^2} 3^i \leq n^2 \cdot 2^{n^2} = 3^{O(n^2)}$$

■

Anhand dieser Laufzeit ist zu sehen, dass ein praktischer Einsatz mit einem realen Roboter nicht sinnvoll ist.

2.2 Robotik

Für alle vorgestellten Navigationsalgorithmen wird immer ein Robotermodell angenommen, das genau die Fähigkeiten hat, die benötigt werden, um den Algorithmus fehlerfrei bis zum Ziel zu führen. Wenn solche Algorithmen auf einen realen Roboter portiert werden, so entstehen weitere Probleme und Fragen, wie z. B. : „Wie erlangt der Roboter Informationen über seinen Standort?“

2.2.1 Dead reckoning - Wo bin ich?

Leonard und Durrant-Whyte (LDW91) haben das Problem der Roboter-Navigation auf drei W-Fragen reduziert:

- *Wo bin ich?*
- *Wo will ich hin?*
- *Wie komme ich dahin?*

Die verschiedenen Verfahren des dead reckoning sollen eine Antwort auf die erste Frage geben. *Dead reckoning* ist ein Kunstwort. Es entstand aus dem Ausdruck „deduced reckoning“ aus der Seefahrt (BEF96). Es bezeichnet eine einfache Rechenvorschrift, um aus dem Kurs und der Geschwindigkeit, die derzeitige Position auszurechnen.

Welche Möglichkeiten gibt es, die Position des Roboters zu bestimmen?

Man kann die Verfahren der Positionsbestimmung grob in zwei Kategorien unterteilen (BEF96).

- Relative Bestimmung der Position
  - Odometrie: Die gebräuchlichste Form der Positionsbestimmung bei einem Roboter. Sie wird weiter unten genauer erläutert, da dies das gewählte Verfahren für diese Diplomarbeit ist.
  - Interne Navigation: Diese Methode nutzt Gyroskope sowie (seltener) Beschleunigungssensoren, um die Rotation und Beschleunigung zu messen und daraus auf die derzeitige Position zu schließen.
- Absolute Bestimmung der Position
  - Aktive Funkfeuer: Der Roboter berechnet seine Position aus drei oder mehr aktiven Funkfeuern (z. B. über Triangulation).
  - Erkennung von künstlichen Landmarken: In der Umgebung werden künstliche Landmarken verteilt, deren Positionen in der Welt bekannt sind. Diese Landmarken sind so konstruiert, dass sie besonders leicht vom Roboter erkannt werden können.
  - Erkennung von natürlichen Landmarken: Besondere Merkmale der Landschaft werden vom Roboter erkannt und als Landmarken benutzt.
  - Internes Modell der Umgebung. Der Roboter bekommt ein Modell der Umgebung mitgeliefert. Er vergleicht die derzeitigen Sensorwerte mit dem Modell um herauszufinden, wo er sich gerade befindet.

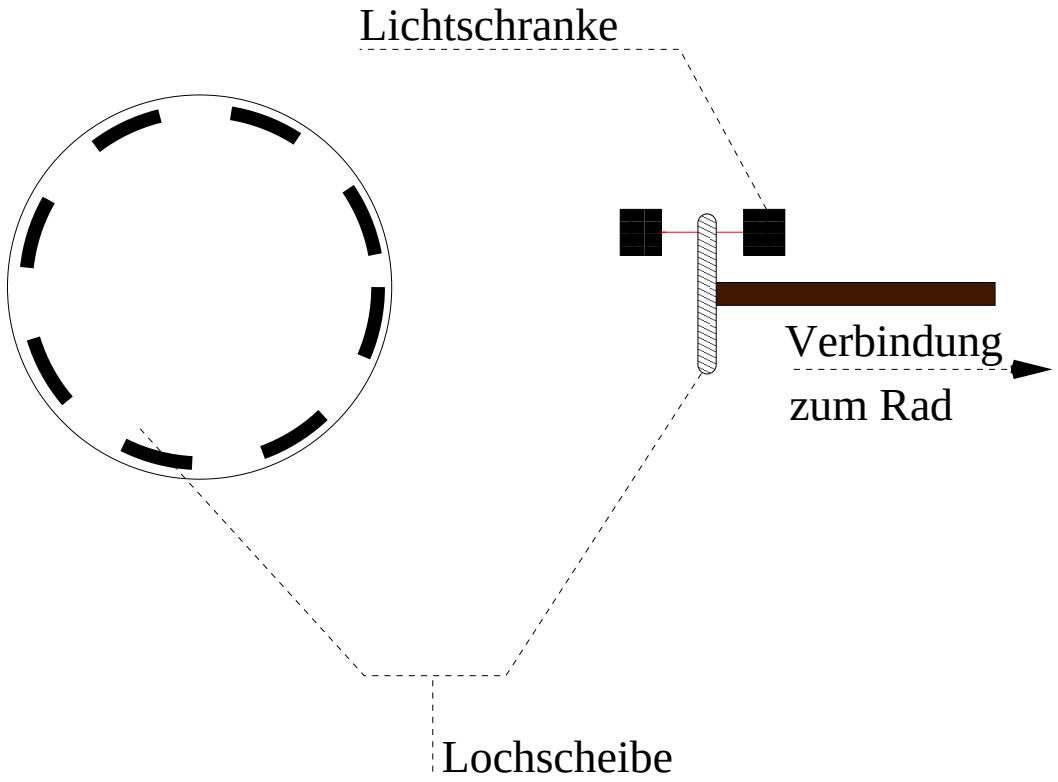

Abbildung 2.6: *Schematischer Aufbau eines optischen Radsensors (optical encoder). Die Scheibe ist mit der Achse des Rades verbunden. Wenn sich das Rad dreht, dreht sich die Scheibe mit. Die Lichtschranke wird in bestimmten Zeitabständen unterbrochen. Aus diesen Unterbrechungen, kann die zurückgelegte Strecke und die Geschwindigkeit berechnet werden.*

Odometrie

Odometrie ist die gebräuchlichste Form eines dead reckoning Verfahrens zur Bestimmung der Position eines Roboters. Die dafür notwendigen Sensorinformationen sind sehr leicht und auch kostengünstig zu bekommen. Wichtig für die Odometrie ist die Messung des Weges, den ein Rad zurückgelegt hat. Dies wird typischerweise über *optische Encoder* gemessen (vgl. Abb 2.6). Diese sind in der einfachsten Form Scheiben, die in gleichmäßigen Abständen am äußeren Rand Löcher haben. Eine Lichtschranke wird so angebracht, dass die Scheibe, wenn sie rotiert, immer wieder diese Lichtschranke unterbricht und dann wieder das Licht durch die Löcher auf den Sensor fallen lässt. Wenn diese Lochscheibe mit dem Motor des Rades fest verbunden wird, so ist

die Anzahl der Impulse der Lichtschranke (jedes Mal, wenn Licht auf den Sensor der Lichtschranke fällt, erzeugt diese einen Impuls) proportional zu der zurückgelegten Strecke.
Mit Hilfe eines solchen Sensors kann die Position des Roboters mit einfachen mathematischen Zusammenhängen berechnet werden. Die folgenden Formeln gelten für das Modell eines Roboters, welches dem Khepera II entspricht (vgl. Abb. 2.7).

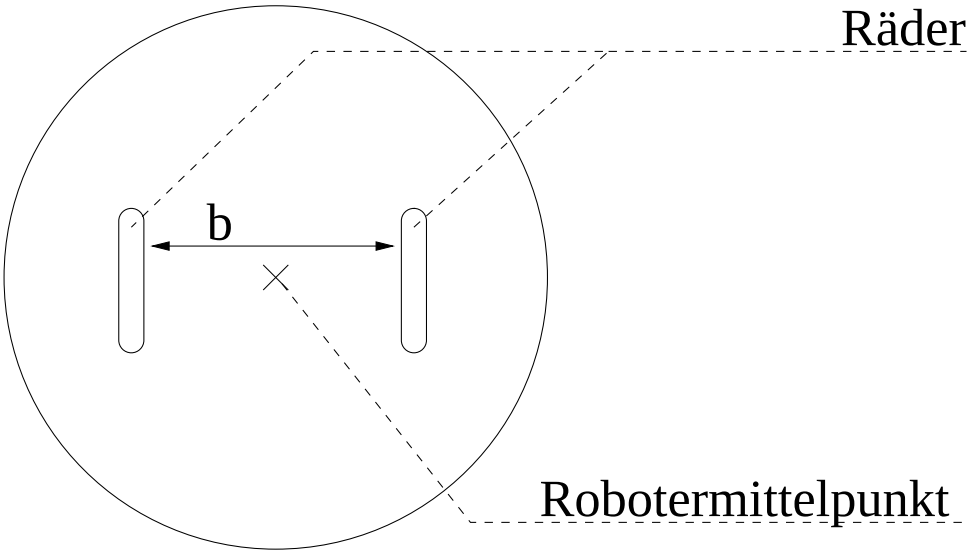

Abbildung 2.7: *Das Modell eines Roboters mit zwei getrennt angetriebenen Rädern. Diese Modellannahme entspricht dem Khepera II.*

Sei D_n der Durchmesser des Rades, C_e die Auflösung des Sensors (in Pulsen pro Umdrehung) und n die Übersetzung zwischen Motor (an dem der Sensor angebracht ist) und Rad.
Ferner sei c_m der Umrechnungsfaktor, der angibt, welcher Strecke ein Puls entspricht.

$$c_m = \pi D_n / n C_e$$

Seien N_L und N_R die gemessene Anzahl an Pulsen seit der letzten Messung für den Motor links bzw. den Motor rechts. Mit Hilfe des Umrechnungsfaktors kann nun die zurückgelegte Strecke des jeweiligen Rades berechnet werden:

$$\Delta U_{L/R} = c_m N_{L/R}$$

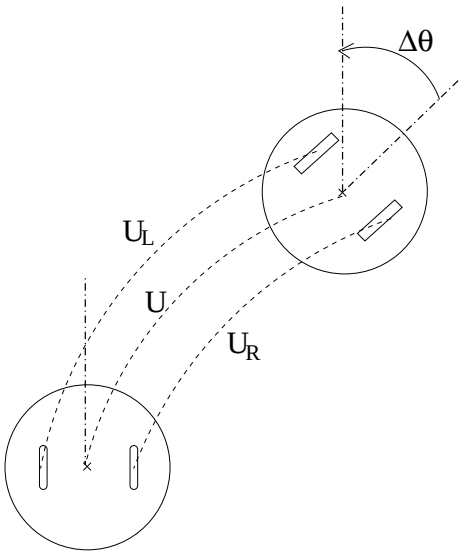

Abbildung 2.8: *Berechnung der Position nach einer bestimmten Zeit. Die Strecken U_L und U_R werden durch Sensoren gemessen. Dann kann $\Delta\Theta$ berechnet werden. Durch Aufsummierung aller Messungen wird die derzeitige Position und Ausrichtung berechnet.*

Die zurückgelegte Strecke des Robotermittelpunktes ist der Mittelwert der zwei Radstrecken:

$$\Delta U = (\Delta U_R + \Delta U_L)/2$$

Die Änderung der Ausrichtung des Roboters ergibt sich durch:

$$\Delta\Theta = (\Delta U_R - \Delta U_L)/b$$

Hierbei ist b der Abstand zwischen den beiden Rädern. Idealerweise wird hier der Abstand zwischen den beiden Punkten der Bodenberührung gemessen.

Die derzeitige relative Ausrichtung des Roboters wird durch Aufsummierung aller Messungen der Ausrichtungen berechnet.

Probleme und Schwierigkeiten der Odometrie

Eine Positionsbestimmung mit Hilfe der Odometrie basiert auf einfachen Gleichungen, welche leicht implementiert werden können. Allerdings basiert Odometrie auch auf der Annahme, dass die Daten, die die Radsensoren liefern, die tatsächliche Radbewegung wiederspiegeln. Die Fehler, die bei der Odometrie gemacht werden, können in zwei Kategorien eingeteilt werden (BEF96). Dies sind einerseits *Systemische Fehler*, d.h. Fehler, die direkt im System auftreten und sich deshalb ständig aufsummieren. Anderseits Fehler, die durch die Umwelt impliziert werden (*nicht-Systemische Fehler*).

Systemische Fehler sind:

- ungleiche Raddurchmesser
- der durchschnittliche Durchmesser des Rades weicht vom angegebenen Durchmesser ab
- der tatsächliche Radabstand weicht vom angegebenen Radabstand ab
- schlecht zentrierte Räder
- endliche Auflösung des Radsensors
- endliche Abtastrate des Radsensors

dem gegenüber stehen die
Nicht-Systemische Fehler:

- unebener Boden
- unerwartete Objekte auf dem Boden
- Schlupf der Räder durch
  - rutschige Oberfläche
  - zu starke Beschleunigung
  - schnelles Drehen (skidding)
  - Kräfte von Außen
  - Kräfte von Innen (Bewegliche Teile im Inneren des Roboters können diesem bei bestimmten Ansteuerungen einen Impuls in unerwünschte Richtung geben)
  - Verlust des Bodenkontaktes

Bei kontrollierten Umgebungen (Spielwelten) überwiegen die systematischen Fehler, da z. B. eine gute Bodenhaftung garantiert werden kann und keine unerwarteten Hindernisse herumliegen. Bei unbekannten und realen Umgebungen überwiegen normalerweise die nicht-systematischen Fehler.
Odometrie ist relativ fehleranfällig und leichte Fehler können sich sehr schnell aufaddieren und dadurch signifikant werden. Deshalb wird meist eine Kombination von Odometrie und eines Verfahrens, welches Landmarken erkennt, benutzt. So können Fehler, die zwischen den Landmarken auftreten, an diesen ausgeglichen werden. Die Position im (kleinen) Bereich zwischen bekannten Landmarken, kann dann recht sicher mittels Odometrie nachverfolgt werden.

2.2.2 Der Roboter - Khepera II

Der Name „Khepera“ kommt aus dem ägyptischen (äquivalent zu Khepri, Kheper, Chepri, Khepra). In altägyptischen Hieroglyphen:

Er bezeichnet einen „göttlichen Mistkäfer“, der in der Vorstellung der alten Ägypter dafür verantwortlich war, die Sonne tagsüber über den Himmel zu schieben. Nachts schob er die Sonne durch die Unterwelt, um am nächsten Tag die Sonne wieder im Osten aufgehen lassen zu können.
Der Roboter „Khepera“ ist eine Entwicklung der K-TEAM Corporation[2], eine schweizer Firma, die sich darauf spezialisiert hat, autonome Roboter für Forschung und Lehre zu entwickeln und zu bauen. Neben dem Khepera Modell, welches das wohl bekannteste der K-TEAM Corporation ist, gibt es noch weitere Modellreihen:

- *Hemisson*: Eine Plattform, die hauptsächlich für Hobbyeinsätze entwickelt wurde. Vom groben Aufbau ähnelt der Hemission dem Khepera, allerdings wurden beim Khepera deutlich höherwertige Materialien und Elektronikbauteile verwendet. Dies zeigt sich auch deutlich im Anschaffungspreis.
- *KOALA*: Der KOALA ist ein mittelgroßer Roboter, der entwickelt wurde, um in realen Umgebungen eingesetzt zu werden. Er ist ein sechsrädriger Roboter, der eine große Anzahl zusätzlicher Ausrüstung tragen kann.

[2](**www.k-team.com**)

Der Khepera selber ist ein relativ kleiner Roboter (Durchmesser etwa 55 mm). Ursprünglich wurde er für die Ecole Polytechnique Fédérale de Lausanne (EPFL) entwickelt. Dort sollte der Khepera eingesetzt werden, um Algorithmen für die Bewegungsplanung in der Praxis zu testen.
Der Khepera entwickelte sich jedoch schnell zu einem Standard in der Robotik, so dass mittlerweile sehr viele Forschungsgruppen und Universitäten mit ihm arbeiten. Das K-Team zeigt auf ihrer Homepage einige Projekte, Forschungsgruppen und Lehrveranstaltungen, in denen mit dem Khepera gearbeitet wurde. Hier können nur einige wenige Arbeiten beispielhaft genannt werden (nicht alle sind auf der Seite des K-Teams zu finden):

- *Mobile Robot Miniaturisation: A Tool for Investigation in Control Algorithms* (MFI94), eine Veröffentlichung, in der der Khepera das erste Mal beschrieben wird und in der zusätzliche Tools vorgestellt werden, mit denen er programmiert werden kann.

- *Collective and Cooperative Group Behaviours: Biologically Inspired Experiments in Robotics* (MM95), eine Veröffentlichung, die sich mit kooperativen Verhalten mehrerer Roboter beschäftigt. Zwei unterschiedliche Ansätze werden hier beschrieben. Einmal ein Schwarmverhalten, welches Kisten aufräumt. Der Effekt des Kistenräumens ist nicht vorprogrammiert, sondern ergibt sich aus einfachen Regeln für jedes Individuum des Schwarms. Der zweite Ansatz ist ein kooperativer Ansatz, um eine Aufgabe zu lösen, welche von einem Roboter alleine nicht zu lösen wäre.

- *Experiments with the Mini-Robot Khepera* (LMR99): Hier finden sich verschiedenste Anwendungen für den Khepera, die auf der ersten Khepera Konferenz 1999 vorgestellt wurden.

- *Golf Playing Khepera* (GKR99), in der ein Khepera mit Hilfe einer Karte und Navigation mit neuronalen Netzen Golf spielt.

Aufbau des Roboters - Die Hardware

Sensorik

Die Details zur Hard- und Software sind hauptsächlich aus (KT02b) entnommen.
Der Khepera II Roboter ist mit 8 Sensoren ausgestattet. Die Sensoren sind wie in Abbildung 2.9 angebracht. Demnach hat der Khepera eine fast 180 Grad Sicht nach vorne und zusätzlich noch Sicht nach hinten.
Die Sensoren des Khepera II sind Infrarot-Distanzsensoren. Diese Sensoren basieren auf einer Messung der Umgebungshelligkeit im Infrarot-Spektrum (*infra-red proximity and ambient light sensors*). Wenn die Umgebungshelligkeit gemessen ist, senden die

Sensoren einen Lichtblitz im Infrarot Bereich aus. Dann wird wieder eine Helligkeitsmessung durchgeführt. Über die Unterschiede zwischen der Umgebungshelligkeit und der Helligkeit bei dem Lichtblitz kann die Entfernung eines Objektes bestimmt werden. Ist ein Objekt nahe am Roboter, wirft es viel Licht des Lichtblitzes zurück und der Unterschied zwischen Umgebungshelligkeit und aktiver Distanzmessungshelligkeit ist groß. Ist kein Objekt vorhanden, wird kein Licht zurückgeworfen, der Unterschied ist nahe Null. Je nach gemessener Lichtstärke ändert sich der Innenwiderstand des Sensors. Es existiert also eine bijektive Abbildung zwischen den Wiederstandswerten und dem Abstand zu einem Objekt. Diese Abbildung ist allerdings nicht linear.
Die Messung der Sensoren wird alle 20 Millisekunden erneut durchgeführt, so dass fünfzig mal in der Sekunde neue Werte vorliegen.

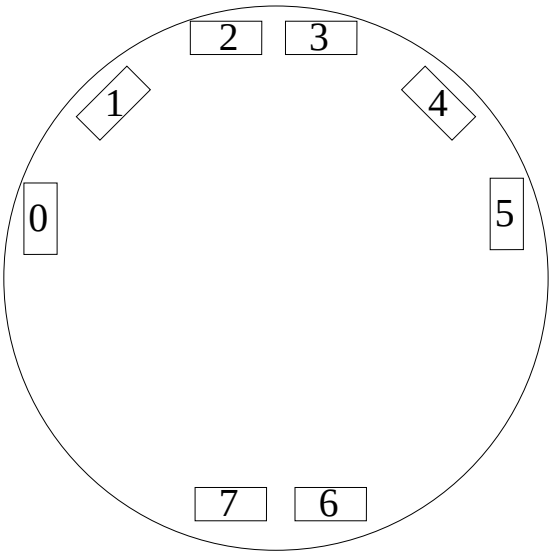

Abbildung 2.9: *Schematische Sensoranordnung des Khepera II*

Nur mit diesen Distanzsensoren hätte sich der Khepera wohl nicht als ein Standard etabliert. Eine der besonderen Fähigkeiten des Kheperas ist die Erweiterung mit Türmen (engl. turrets). Diese Türme werden oben auf den Khepera gesteckt und von diesem mit Strom versorgt. Türme können sehr unterschiedliche Funktionsweisen haben. So werden vom K-Team selber verschiedene Erweiterungen angeboten. Einerseits gibt es den so genannten „Gripper“. Zum einen ein Greifer, der es dem Khepera ermöglicht, seine Umwelt zu manipulieren und z. B. kleine Transportaufgaben auszuführen. Ebenso werden verschiedene Türme mit Kameras angeboten. Dies geht von einfachen Linearkameras bis hin zu komplexen Matrix-Farb-Kameras mit eigenen Chips für eine Bildverarbeitung.

Den Funktionen der Türme sind jedoch im Prinzip keine Beschränkungen gesetzt (sieht man einmal vom Gewicht ab, welches der Khepera bewältigen kann). Je nach Konstruktion des Turmes können mehrere gleichzeitig übereinander am Khepera angebracht werden.
Die Türme werden vom Khepera über das so genannte *KNET*-Protokoll angesprochen. Das Protokoll ist ein Master-Slave-Netzwerk-Protokoll. Hierbei sind die Türme die Slaves und der Khepera der Master. Die Kommunikation wird immer vom Khepera aus angestoßen. Um die Türme ansprechen zu können, hat jeder Turm eine eigene ID. Wenn auf das Netzwerk vom Khepera aus Daten gelegt werden, so prüft jeder Turm, ob diese Daten mit seiner ID versehen sind. Falls ja, wird der Turm auf Empfang geschaltet. Falls nein, schaltet der Turm nicht auf Empfang. Weitere Details und die Pinbelegungen des Khepera-Turm-Anschlusses sind in (Fra00) zu finden.

Motorik
Der Khepera II wird von 2 Servomotoren (*DC brushed servo motors*) angetrieben. Jedes der beiden Räder wird separat angetrieben. An jedem Motor ist zusätzlich ein Sensor angebracht, der die Umdrehung misst (wie in Abbildung 2.6 angegeben). Nach Herstellerangaben entsprechen 12 Pulse dieses Sensors (12 „Motorticks“) ein Millimeter der Roboterbewegung.
Es gibt beim Khepera II zwei Möglichkeiten die Motoren anzusteuern:

1. Direkte Ansteuerung der Geschwindigkeit
2. Angabe einer relativen Position

Beide Ansteuerungen sind im Khepera II mit Regelkreisen realisiert.
Bei der direkten Ansteuerung wird jedem Motor eine Geschwindigkeit zugewiesen. Ein positiver Wert entspricht hierbei einer Fahrt nach vorne, eine negativer Wert einer Fahrt rückwärts. Der Wert Null bedeutet, dass der Motor stehen bleibt. Mit diesen Werten können beliebige Kurven und Geradeausfahrten realisiert werden. Ebenso ist eine Drehung auf der Stelle möglich. Der Regelkreis registriert mit Hilfe des Motor-Encoders die Geschwindigkeit der Motoren. Wenn diese Geschwindigkeit nicht mit der gesetzten Geschwindigkeit übereinstimmt, wird nachgeregelt (vgl. Abb. 2.10).
In der zweiten Ansteuerung (*durch Angabe einer relativen Position*) gibt der Benutzer den Motoren jeweils einen Wert für den Motor Encoder vor. Da ein Puls des Motor-Encoders etwa 1/12 Millimeter entspricht und die Werte des Motor-Encoders beim Start des Roboters auf Null gesetzt sind, kann der Roboter durch Angabe von Encoder-Werten relativ zum Startpunkt des Roboters gesteuert werden. Der Regelkreis braucht als zusätzliche Eingabe eine Trajektorie, die von der aktuellen Position zur vorgegebenen führt (vgl. Abb. 2.11). Zusätzlich sorgt der Regelkreis dafür, dass der Roboter „weich“ anfährt und auch „weich“ abbremst.

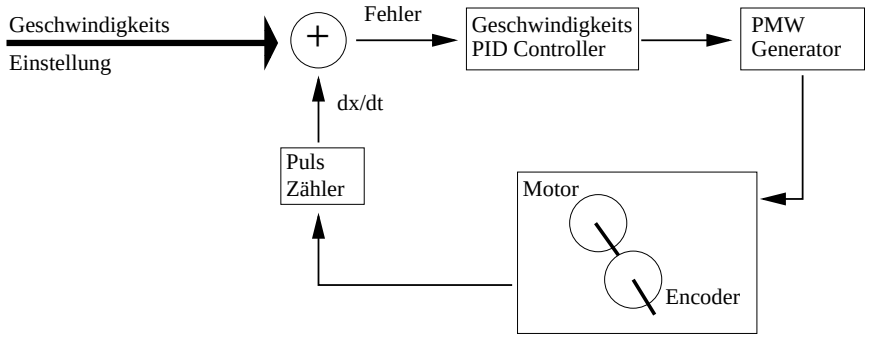

Abbildung 2.10: *Schema des Regelkreises für die Geschwindigkeitssteuerung. Anhand der Daten des optical encoders wird die derzeitige Geschwindigkeit bestimmt. Diese wird mit der Vorgabe des Benutzers verglichen. Der Fehlerwert, der sich daraus ergibt, sorgt dafür, dass die Motorenansteuerung sich an die Benutzervorgaben annähert. Das Verhalten des Regelkreises ist das Verhalten eines PID-Regelkreis.*

Der Mikroprozessor
Der Khepera II ist mit einer Motorola 68311 CPU ausgestattet. Diese CPU hat eine 32-Bit-Architektur. Im Khepera II ist sie mit 25 MHz getaktet und verfügt über 512kb internen Speicher und einen zusätzlichen 512kb Flash-Speicher, der über einen seriellen Port programmiert werden kann.

Steuerung des Roboters - Die Software

Das Kommunikationsprotokoll
Das serielle Kommunikationsprotokoll ist dafür gedacht, über einen RS232 Port alle Funktionen des Kheperas anzusteuern.
Die Übertragung hat folgende Parameter:

- 8-Bit-Übertragung
- 1 Start-Bit
- 1 Stopp-Bit
- keine Parität

Die Kommunikation zwischen Roboter und Rechner findet über die erweiterte ASCII-Codierung statt.

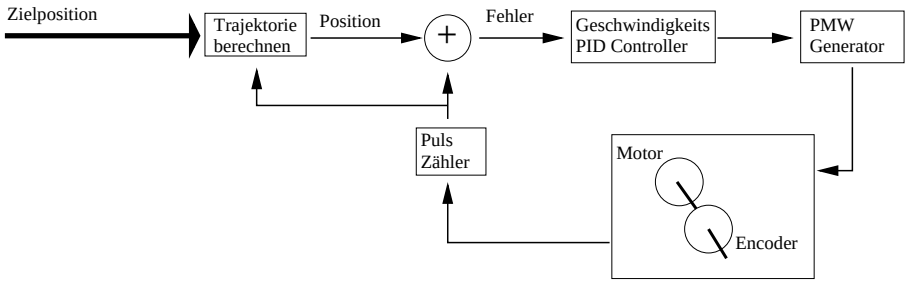

Abbildung 2.11: *Schema des Regelkreises für die Positionsbestimmung. Der Benutzer gibt je einen Wert für beide Motor-Encoder vor. Es wird eine gleichförmige Bewegung für jeden Motor berechnet, so dass beide Motor-Encoder gleichzeitig die für sie gesetzten Werte erreichen (Berechnet in der Trajektorie-Box). Auch hier ist der Regler ein PID-Regler. Nicht in diesem Schema enthalten ist, dass sowohl das Anfahren, wie auch das Abbremsen nicht abrupt geschieht, sondern weich gesteuert wird.*

Die Kommunikation muss immer über den Rechner angestoßen werden. Die Befehle werden als Großbuchstaben, wenn nötig gefolgt von Parametern, an den Roboter gesendet. Der Roboter reagiert dem Befehl entsprechend und sendet eine Antwort mit Kleinbuchstaben.

Ein einfaches Beispiel: D,4,-5
Befehl: D
Parameter: speed_linker_motor, speed_rechter_motor
Effekt: Setzt die Geschwindigkeit der beiden Motoren
Antwort des Roboters: d¶

Der Krosscompiler
Der Krosscompiler ist ein **K**hepera C**rosscompiler**, der C-Programme in den Befehlssatz des Motorola-Prozessors übersetzt.
So kann der Roboter programmiert werden, ohne dass der Entwickler genaustes über die Hardware informiert ist.
Die Informationen zum Programmieren des Khepera II sind entnommen aus (KT02a). Dort finden sich auch weitere Informationen und Beispielprogramme.

Eine wichtige Fähigkeit des Chipsatzes des Khepera II ist die Multi-Task-Fähigkeit. Diese Fähigkeit ermöglicht es, verschiedene Prozesse interagieren zu lassen, verlangt aber vom Programmierer erhöhte Aufmerksamkeit. So können Prozesse die gleichzei-

tig auf die Motorsteuerung zugreifen, ungewollte Seiteneffekte bis hin zu einem Reset des Roboters, erzeugen. Also muss bei gemeinsamer Nutzung von Ressourcen darauf geachtet werden, dass nicht gleichzeitig verschiedene Prozesse darauf zugreifen. So kann z. B. eine wechselseitige Nutzung von Ressourcen zu Deadlocks führen. Dies kann mit bekannten Methoden, wie z. B. Semaphoren verhindert werden.
Der Taskmanager unterstützt folgende Funktionen:

- Anmelden von Tasks
- Abmelden (kill) von Tasks
- Tasks warten lassen, bis...
  - ... eine bestimmte Zeit vorbei ist
  - ... ein Event eintritt, welches in einem anderen Task erzeugt wurde
  - ... ein externes Event eintritt, welches durch Peripherie erzeugt wurde

Die Motoren:
Die Motoren können, bedingt durch die zwei unterschiedlichen Regelkreise, auf zwei Arten programmiert werden:

- *Speed Control:* Hier wird den Motoren eine bestimmte Geschwindigkeit zugewiesen. Die dazugehörigen Befehle sind:

mot_config_speed_1m():	Setzt die Parameter für den Regelkreis
mot_new_speed_1m():	Setzt die Geschwindigkeit für einen Motor
mot_new_speed_2m():	Setzt die Geschwindigkeit für zwei Motoren
mot_get_speed():	Gibt die aktuellen Motorwerte zurück

- *Position Control:* Hier wird der Roboter dadurch gesteuert, dass man die Werte angibt, die die beiden Motor-Encoder annehmen sollen. Der Regelkreis steuert die Motoren so, dass der Roboter diese Zielkoordinaten erreicht. Die dazugehörigen Befehle sind:

mot_config_position_1m():	Setzt die Parameter für den Regelkreis
mot_config_profil_1m():	Setzt die Parameter für die Geschwindigkeitsregelung
mot_get_position():	Gibt die aktuelle Roboterposition aus
mot_put_sensor_1m():	Setzt die Werte eines Motor-Encoders
mot_put_sensor_2m():	Setzt die Werte beider Motor-Encoder
mot_new_position_1m():	Setzt das Ziel für einen Motor-Encoder
mot_new_position_2m():	Setzt das Ziel für beide Motor-Encoder

Die Sensoren können zwei verschiedene Messungen vornehmen:

- *Messung der Umgebungshelligkeit (ambient light):* Eine rein passive Messung der derzeitigen Helligkeit im messbaren Bereich der Sensoren. Der Befehl dafür ist:

  sens_get_ambient_value(): Gibt einen Wert für die Umgebungshelligkeit wieder

- *Distanzmessung (reflected light):* Eine aktive Messung. Es wird eine Helligkeitsmessung bei aktiver Beleuchtung der Szene durchgeführt. Durch die Differenz zwischen Ausleuchtung und Umgebungshelligkeit kann die Distanz zu einem Objekt berechnet werden. Der Befehl dafür ist:

  sens_get_reflected_value(): Gibt einen Wert für die Distanzmessung wieder

Um mit den Türmen des Kheperas zu kommunizieren sind folgende Befehle besonders wichtig:

msg_snd_rec_message ():	Schickt eine Nachricht an einen Turm und empfängt die Antwort
RESERVE_MSG ()	Reserviert das Netzwerk für einen bestimmten Turm
RELEASE_MSG ()	Gibt das Netzwerk wieder frei

Weiter Befehle sind sowohl in (KT02a) wie auch in der Beschreibung des Khepera II Bios zu finden.

Kapitel 3

Entwicklung des Einbahnstraßen-Pledge

In Kapitel 2 - Grundlagen - sind Strategien zur Navigation in einfachen Labyrinthen betrachtet worden. Durch Einführung weiterer Strukturen, sind jedoch beliebig komplizierte Labyrinthe denkbar. So können Labyrinthe existieren in denen es zum Beispiel nicht egal ist, aus welcher Richtung man kommt, oder auch Labyrinthe die im Laufe der Zeit ihr Aussehen verändern können.
In dieser Diplomarbeit werden Labyrinthe betrachtet, in denen Einbahnstraßen vorkommen (vgl. Abb 3.1).

3.1 Labyrinthe mit Einbahnstraßen

Definition 3.1*: Einbahnstraße*
Eine Einbahnstraße ist eine Linie im Labyrinth, welche vom Roboter nur in eine Richtung überfahren werden darf.
Sie wird gegeben durch ein Liniensegment $\overline{pq}$ mit den Punkten p, q auf dem Rand der polygonalen Ketten und einem Richtungsvektor $\vec{d}$, der orthogonal zu $\overline{pq}$ steht. Der Vektor $\vec{d}$ gibt die Richtung an, in der die Einbahnstraße überfahren werden darf.

Als Einbahnstraße wird die Verbindung von zwei Punkten p und q und einer Richtung $\vec{d}$ bezeichnet. p und q liegen auf den polygonalen Ketten, die die Wände des Labyrinthes bilden. Hierbei darf $\overline{pq}$ kein Hindernis schneiden, oder auf dem Rand von Hindernissen verlaufen (vgl. Abb. 3.2). Ebenso darf der Schnitt zweier Einbahnstraßen maximal die jeweiligen Randpunkte enthalten.
Es muss also gelten:
Seien $\overline{pq}$ und $\overline{lm}$ zwei Einbahnstraßen.

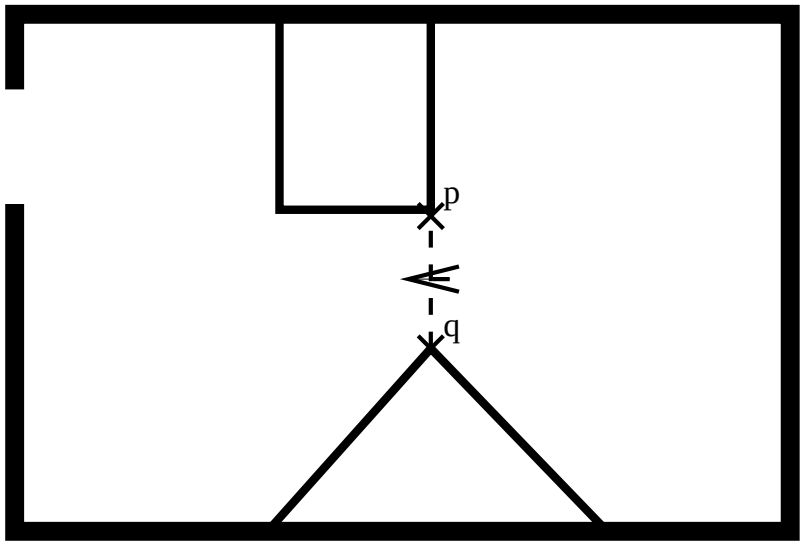

Abbildung 3.1: *Ein Labyrinth mit einer Einbahnstraße. Die Pfeilspitze gibt an, in welche Richtung die Einbahnstraße durchfahren werden kann.*

$$\overline{pq}\backslash\{p,q\} \cap \overline{lm}\backslash\{l,m\} = \emptyset\,.$$

Die Richtung $\vec{d}$ ist orthogonal zu $\vec{pq}$ und gibt an, in welche Richtung der Roboter $\overline{pq}$ überschreiten darf.
Sei $\vec{v}$ die Fahrtrichtung (*Blickrichtung*) des Roboters, so gilt:

$$\vec{v} = \lambda_1 \vec{d} + \lambda_2 \vec{pq}$$

Wenn $\lambda_1 > 0$ so darf der Roboter $\overline{pq}$ überschreiten,
wenn $\lambda_1 \leq 0$ so darf der Roboter $\overline{pq}$ nicht überschreiten.
Anders ausgedrückt: Solange der Winkel zwischen $\vec{d}$ und $\vec{v}$ (in mathematischer Drehrichtung) zwischen $90°$ und $270°$ liegt, darf der Roboter $\overline{pq}$ nicht überschreiten.

Eine Einbahnstraße ist nach dieser Definition eine Linie, die nur in eine Richtung überschritten werden darf und keine Ausdehnung in der Breite hat.
Das Konstrukt der Einbahnstraße ist z.B. durch Verkehrsregeln (Straßenverkehr) oder durch Oberflächenstrukturen zu motivieren. So kann z.B. eine kleine Rampe im Labyrinth installiert sein, deren Abbruchkante vom Roboter nicht überwunden werden kann (vgl. Abb. 3.3). Ebenso kann die Navigation in Labyrinthen mit Einbahnstraßen für einen Serviceroboter interessant sein. Angenommen, solch ein Roboter hat keine

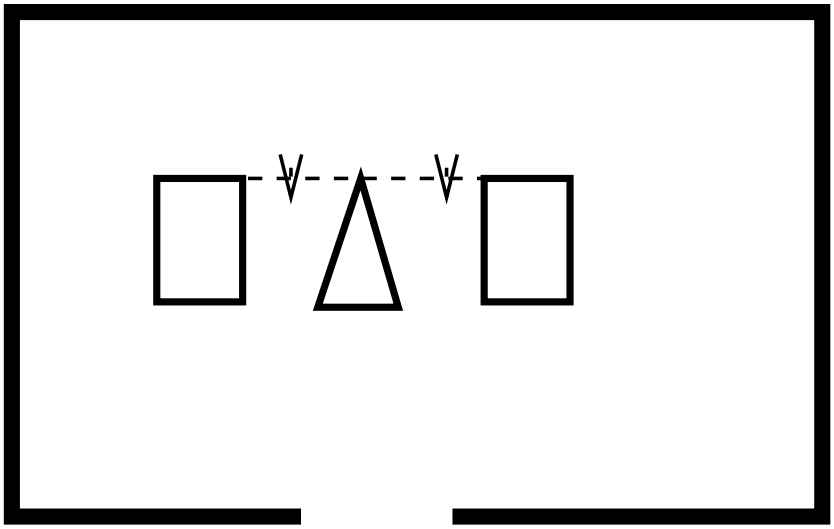

Abbildung 3.2: *Dies sind zwei verschiedene Einbahnstraßen. Nach Definition gelten diese Einbahnstraßen nicht als eine, da ein Punkt des Hindernisses nur am Rand einer Einbahnstraße liegen darf.*

Aktuatoren, wie z.B. Greifarme (als ein Beispiel der Roboter RHINO (BBC$^+$95)). Er soll sich aber dennoch innerhalb eines Gebäudes bewegen. Türen die sich durch einfaches drücken öffnen lassen, kann er passieren. Dieselbe Tür ist für ihn in die andere Richtung allerdings unpassierbar, da er nicht in der Lage ist, die Tür zu ziehen.

Definition 3.2*: Zustände einer Einbahnstraße*
Ein Roboter kann sich von zwei Seiten einer Einbahnstraße nähern:

- Verbotene Seite: *Der Roboter registriert die Einbahnstraße als Wand.*
- Durchfahrbare Seite: *Der Roboter registriert, dass eine Einbahnstraße vorhanden ist. Sie kann von dieser Seite durchfahren werden.*
  - *Eine Einbahnstraße gilt als* geschlossen*, wenn diese Einbahnstraße vom Roboter von der durchfahrbaren Seite* ***nicht durchfahren*** *werden soll. Der Roboter registriert die Einbahnstraße, interpretiert sie aber als Wand.*
  - *Eine Einbahnstraße gilt als* offen*, wenn diese Einbahnstraße vom Roboter von der durchfahrbaren Seite* ***durchfahren*** *werden soll. Der Roboter registriert die Einbahnstraße, interpretiert sie als freie Fläche.*

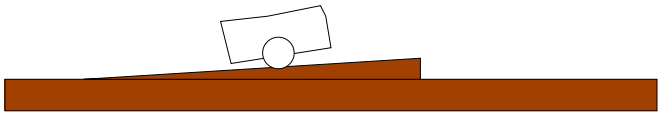

Abbildung 3.3: *Eine Rampe als Realisierung einer Einbahnstraße. Der Roboter kann von der linken Seite auf die rechte Seite gelangen, nicht aber umgekehrt.*

Wenn einem bestehenden Labyrinth Einbahnstraßen hinzugefügt werden, so kann dieses Labyrinth möglicherweise nicht mehr von jeder Position aus verlassen werden.

Definition 3.3*: Faires Labyrinth*
Ein Labyrinth heißt fair, genau dann, wenn:
$\forall p \in C_{frei} \exists$ *Weg von p zum Ausgang.*

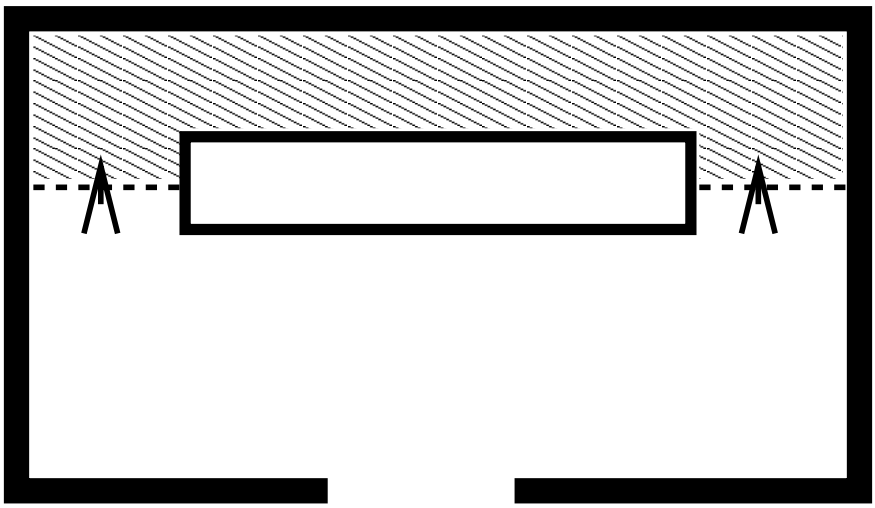

Abbildung 3.4: *Beispiel für ein unfaires Labyrinth. Der Roboter kann in den schraffierten Bereich einfahren, ihn jedoch nicht mehr verlassen.*

Es können also durch Einbahnstraßen *Gebiete* entstehen, aus denen der Roboter nicht mehr entkommen kann (vgl. Abb. 3.4).

Definition 3.4*: Gebiet eines Labyrinths*
Gegeben: Ein Labyrinth L_{ein} mit N Einbahnstraßen.
Ein Gebiet G ist die maximale Menge aller Punkte aus C_{frei}, so dass gilt:
$\forall p_i, p_j \in G$ existiert ein Weg von p_i nach p_j, der weder eine Einbahnstraße noch ein Hindernis schneidet.
Die Punkte, welche auf der Einbahnstraße liegen, werden dem Gebiet zugerechnet, in welches die Einbahnstraße führt.

Aus dieser Definition ergeben sich für Gebiete folgende Eigenschaften:

- $G_i \cap G_j = \emptyset : \forall i \neq j$
- $\bigcup_{i=1}^{n} G_i = C_{\text{frei}}; n =$ *Anzahl Gebiete*

Wie in Abbildung 3.5 zu sehen ist, erzeugen nicht alle Einbahnstraßen ein neues Gebiet oder tragen zu einem Gebiet bei.
Die Aufteilung eines Labyrinthes in seine verschiedenen Gebiete erweist sich als sehr nützlich, da die einzelnen Gebiete als Einbahnstraßen-freie Labyrinthe betrachtet werden können. Einbahnstraßen, die nicht zu einem Gebiet beitragen, d.h. innerhalb eines Gebietes liegen, können als Wände betrachtet werden. Beide Seiten können erreicht werden, ohne diese oder eine andere Einbahnstraßen durchschreiten zu müssen. Ansonsten wären diese Einbahnstraßen Teil einer Abgrenzung eines Gebietes.

Im Weiteren werden nur noch faire Labyrinthe betrachtet. Dies ist notwendig, da die Problemstellung, aus dem Labyrinth zu entkommen, ein Onlineproblem darstellt. Dies bedeutet, dass die Informationen, die notwendig sind, um das Labyrinth zu verlassen, zur Laufzeit gesammelt werden. Das Finden eines unfairen Gebiets erfordert allerdings, unter bestimmten Bedingungen das Einfahren in dieses Gebiet. Selbst mit aufwendigen Verfahren, wie z.B. dem Kartographieren des Labyrinthes, kann es Gebiete geben, die nicht sicher als „unfaires Gebiet“ identifiziert werden können. Nur Gebiete, die komplett umfahren werden können, können auf eine solche Sackgasse untersucht werden. Gebiete die nicht komplett umfahren werden können, könnten den Ausgang enthalten, so das dieses Gebiet kein „unfaires“ ist (vergl. 3.6). Nach dem Einfahren in eine Sackgasse befindet sich der Roboter allerdings in einer ausweglosen Situation. Der Wegfindungsalgorithmus befindet sich in einem Zustand, der keine Lösung („finde den Ausgang“) mehr zulässt.

Um eine bessere Vorstellung dafür zu bekommen, was es heißt in einem Labyrinth mit Einbahnstraßen zu navigieren, werden zuerst die Wege betrachtet die ein Robo-

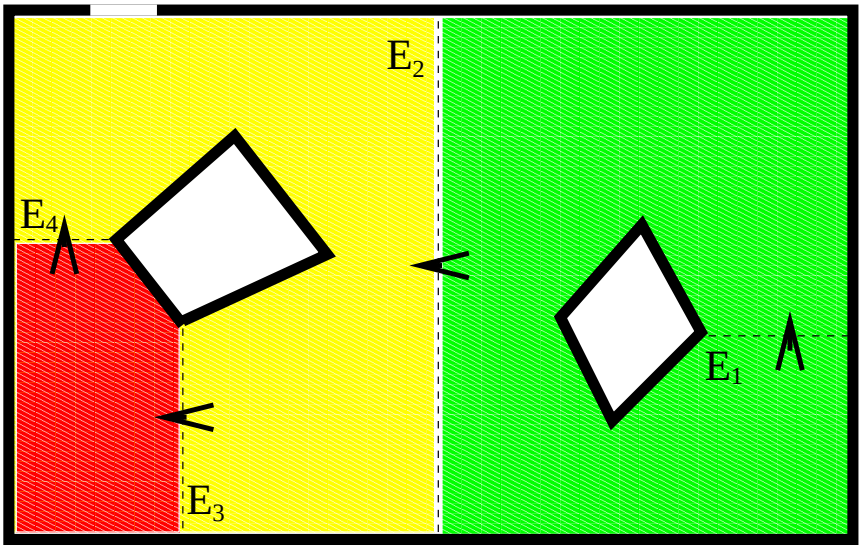

Abbildung 3.5: *Ein Labyrinth mit Einbahnstraßen. Die Gebiete sind farbig unterlegt. E_1 erzeugt kein neues Gebiet, da es Punkte in C_{frei} gibt, von denen aus beide Seiten von E_1 erreicht werden können, ohne eine Einbahnstraße zu überfahren. Für E_2 bis E_4 existieren keine solche Punkte. Sie tragen zu Gebieten bei.*

ter zurück legt. Hierbei wird, wie auch bei den Problem die der Pledge-Algorithmus löst, angenommen, dass der Roboter irgendwo im Labyrinth startet und der Ausgang außen ist, der Roboter also seine Aufgabe gelöst hat, wenn er sich beliebig weit vom Labyrinth lösen kann.

Lemma 3.5*: Zu jedem Labyrinth und jedem Startpunkt existiert jeweils mindestens ein Ausweg, der jede Einbahnstraße maximal einmal durchfährt.*

Beweis. Gegeben ist ein Labyrinth L_{ein} mit Einbahnstraßen. Es seien alle Einbahnstraßen die nicht zu einem Gebiet beitragen, geschlossen (also Wände). Sei G_1 das Gebiet in dem der Ausgang von L_{ein} liegt. Sei G_n das Gebiet[1] in dem der Startpunkt des Roboters liegt.
Es existiert eine Kette von Gebieten G_2 bis G_{n-1}, die von G_1 bis G_n führt. Entweder führt eine Einbahnstraße nicht in ein anderes Gebiet, dann muss sie nicht überschritten werden. Oder sie führt in ein anderes Gebiet, dann muss sie maximal einmal überschritten werden. Angenommen, auf dem Weg müsste die Einbahnstraße zwei oder

[1] Diese Gebiete sind eindeutig

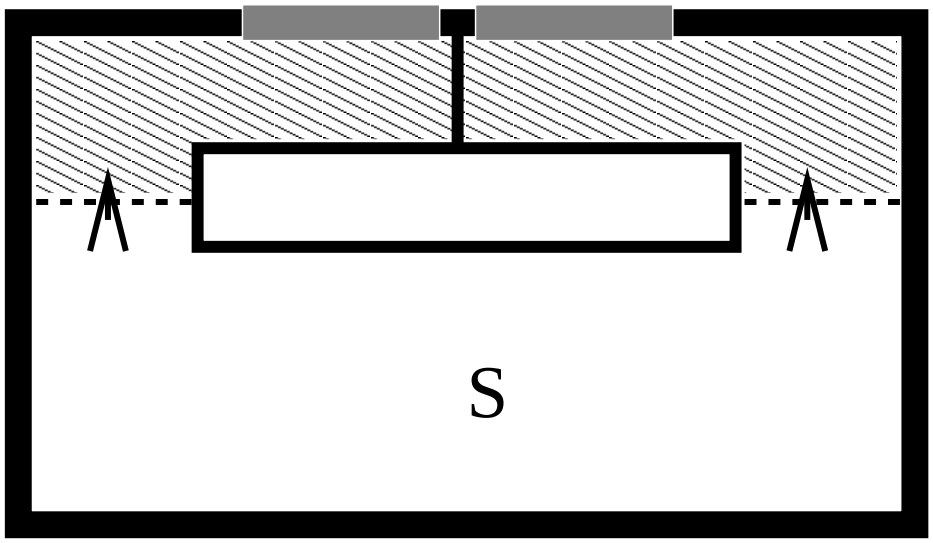

Abbildung 3.6: *Dieses Labyrinth ist ein Beispiel dafür dass Labyrinthe online nicht auf Fairness untersucht werden kann. Vom Startpunkt S aus kann nicht entschieden werden, welches der schraffierten Gebiete eine „Falle“ ist. Das Labyrinth ist lösbar, wenn eine der beiden grauen Balken zu einem Ausgang wird. Es wird jeweils der graue Balken zum Ausgang, der nicht in dem Gebiet liegt, in dass der Roboter als erstes einfährt.*

mehrmals überschritten werden. Dann würde der Weg einen Kreis beschreiben, der abgekürzt werden könnte. Deshalb muss jede Einbahnstraße nur einmal durchlaufen werden. ■

In dieser Arbeit werden Algorithmen vorgestellt, die einerseits jedes faire Labyrinth mit Einbahnstraßen verlassen können, andererseits sollen diese Algorithmen, analog zum Pledge-Algorithmus, möglichst wenig Daten speichern müssen.
Als erster Ansatz bietet sich eine Erweiterung des Pledge-Algorithmus an. Hierbei stellt sich allerdings die Frage, ob es ausreicht, eine Änderung des Pledge-Algorithmus vorzunehmen oder ob eine vergleichsweise komplizierte Steuerung notwendig ist.
Der Unterschied zwischen einem Labyrinth mit Einbahnstraßen und einem einfachen Labyrinth liegt für einen Pledge-Algorithmus in der Notwendigkeit, Entscheidungen zu treffen. Entscheidungen sind hier so zu verstehen, dass es zwei gleichberechtigte Möglichkeiten gibt: jede Einbahnstraße, die der Roboter durchlaufen dürfte, kann, muss aber nicht, durchfahren werden. In Abbildung 3.7 ist sehen, dass, wenn die Einbahnstraße *geschlossen* wäre, der Pledge-Algorithmus einen Weg finden würde.

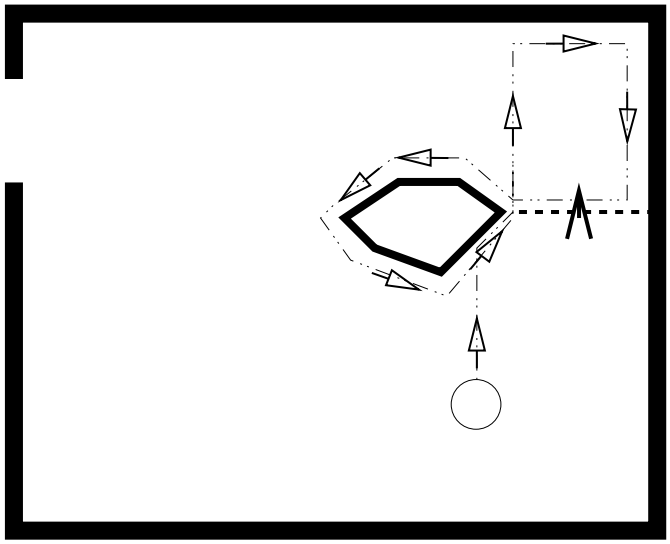

Abbildung 3.7: *Ein Beispiellabyrinth mit Einbahnstraße aus dem der Pledge-Algorithmus nicht herausfindet.*

Am Beispiel 3.7 ist zu sehen, dass sich Wege, die ein Pledge gesteuerter Roboter in einem Labyrinth mit Einbahnstraßen fahren kann, von den Wegen unterscheidet, die der Pledge in einem Labyrinth ohne Einbahnstraßen nimmt. In Kapitel 2 wurde gezeigt, dass der Weg des Pledge-Algorithmus keine Selbstschnitte aufweisen kann. Schon Abbildung 3.7 zeigt, dass dies in Labyrinthen mit Einbahnstraßen nicht mehr gilt.

Lemma 3.6*: Angenommen, der Roboter wird mit einem Pledge-Algorithmus gesteuert. Die Einbahnstraßen sind vorkonfiguriert auf geschlossen oder offen gesetzt und ändern sich während der Fahrt nicht.*
Seien p und q zwei gerade Teilstücke des Weges des Roboters, die sich schneiden. So muß entweder p oder q durch eine freie Bewegung entstanden sein.

Beweis. Ein Selbstschnitt kann nicht durch zwei freie Bewegungen entstanden sein. Wären p und q beide durch freie Bewegung erzeugt worden, so wären p und q parallel zueinander und könnten sich nicht kreuzen.
Angenommen p und q sind beide durch eine „nicht freie“ Bewegung entstanden. Dies bedeutet, der Roboter ist sowohl bei p wie auch bei q der Wand auf der linken Seite gefolgt. Wenn sich p und q kreuzen, müssen sich auch die Wände kreuzen, denen

der Roboter gefolgt ist. Zusätzlich müsste der Roboter durch die Wand gegangen sein. Da dies nicht möglich ist, muss an dieser Stelle eine Einbahnstraße sein. Die ist von der eine Seite durchfahrbar und von der anderen Seite als Wand zu sehen. Dies muss an beiden Wänden der Fall sein. Alle Möglichkeiten, wie solch ein Konstrukt aussehen kann, sind in Abbildung 3.8 zu sehen. Die Möglichkeiten, die überhaupt nur vom Pledge-Algorithmus gefahren werden können, bedingen, dass die Konfiguration der Einbahnstraßen währen der Fahrt geändert wird. → Widerspruch.

■

Das es Selbstschnitte im Weg des Roboters geben kann, ist für die weitere Arbeit von entscheidender Bedeutung. Im Gegensatz zum Pledge-Algorithmus kann nun nicht mehr angenommen werden, dass sich der Winkelzählerwert des Roboters bei einer geschlossenen Kurve entweder um $-2 \cdot \pi$ oder $2 \cdot \pi$ verändert hat. Hier ist wider Abbildung 3.7 ein Beispiel. Auf der Endlosschleife ändert sich der Winkelzähler an einem bestimmten Punkt, betrachtet über die Zeit, nicht. Wichtig für die weitere Betrachtung ist die Frage, was es für Selbstschnitte geben kann. Es wird hier bei der Betrachtung davon ausgegangen, dass sich während des Weges des Roboters der Status der Einbahnstraßen („geschlossen“ oder „offen“) nicht ändert.
Die Betrachtung, wie viele Selbstschnitte es in einem geschlossen Weg des Pledge-Algorithmus in einem Labyrinth mit Einbahnstraßen geben kann, ist für die Entwicklung eines Algorithmus von Bedeutung.

Lemma 3.7*: Wenn ein Roboter, der mit dem Pledge-Algorithmus durch ein Labyrinth mit Einbahnstraßen*[1] *gesteuert wird, in eine Endlosschleife gerät, so hat diese Endlosschleife maximal einen Selbstschnitt.*

Beweis. Dass eine Endlosschleife einen Selbstschnitt hat, zeigt zum Beispiel Abbildung 3.7. Angenommen es existieren zwei Selbstschnitte. Wie in Lemma 3.6 gezeigt, muss an jedem Selbstschnitt eine der beiden beteiligten Teilstücke aus einer freien Bewegung entstehen. Dies bedeutet bei zwei Selbstschnitten, dass sie parallel und in die gleiche Richtung gehen müssen. Es bedeutet weiterhin, dass der Winkelzählerwert an einem Punkt in der gesamten Endlosschleife über die Zeit nicht kleiner werden darf, da es sonst nach einigen Durchläufen keine freien Bewegungen mehr gibt. Dies ist ein Widerspruch zu der Annahme einer endlosen Schleife mit Selbstschnitten.
Es muss also bei einem Umlauf genau so viel vom Winkelzähler abgezogen wie aufaddiert werden. Dies geht nur mit einer ungeraden Anzahl von Selbstschnitten. In Abildung 3.9 ist so eine Endloskurve mit drei Selbstschnitte gezeigt. Hierbei sind die parallelen Strecken, die die freien Bewegungen darstellen, nicht in die gleichen Richtung durchfahrbar.

[1] Diese Einbahnstraßen stehen auf geschlossen oder offen, ohne während der Fahrt geändert zu werden.

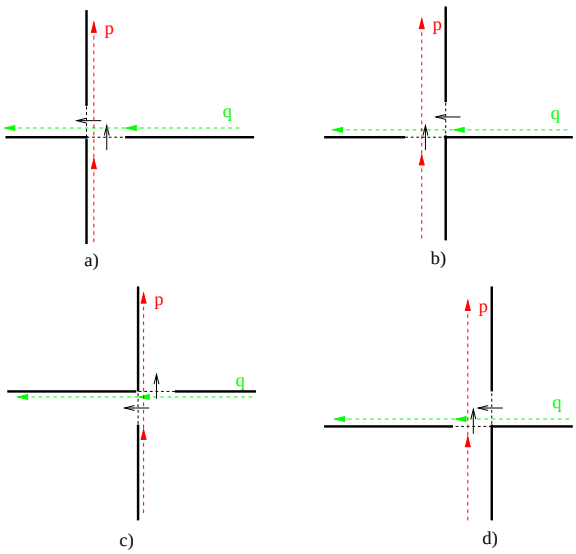

Abbildung 3.8: *Alle Möglichkeiten, wie sich zwei nicht freie Bewegungen kreuzen können. Keine dieser Möglichkeiten kann jedoch von einem Pledge-Algorithmus gefahren werden, wenn die Einbahnstraßen während der Fahrt nicht neu konfiguriert werden. Fall a) müßte nachdem Segment p gefahren wurde, die senkrechte Einbahnstraße umgestellt werden, so das Segment q durch diese fahren kann. Fall b) und Fall c) kann nicht von einem Pledge-Algorithmus gefahren werden, da die Wand bei den beiden Segmenten auf unterschiedlichen Seiten liegt. In Fall d) müßte die waagerechte Einbahnstraße für das Segment p anders konfiguriert sein, als für Segment q.*

Es existieren Konstrukte, in denen die drei freien Bewegungen, die die Selbstschnitten erzeugen, sowohl parallel liegen, wie auch in gleiche Richtung führen. Hierbei kommt allerdings eine andere Eigenschaft des Weges eines Pledge Algorithmus zum tragen. Am Ende einer freien Bewegung dreht sich der Roboter immer in die gleiche Richtung. In dieser Diplomarbeit ist diese, nach Vereinbarung, eine Drehung nach rechts. Jede freie Bewegung in die richtige Richtung erzeugt also zwangsläufig eine Schleife, die den Winkelzähler um 2π verkleinert. Eine geschlossene Kurve, in der der Roboter an jeder Stelle bei jedem Durchlauf den gleichen Winkelzähler hat, ist so also nicht zu konstruieren.
Jede Anzahl von Selbstschnitten größer eins, ist auf diese Fälle zurückzuführen. ■

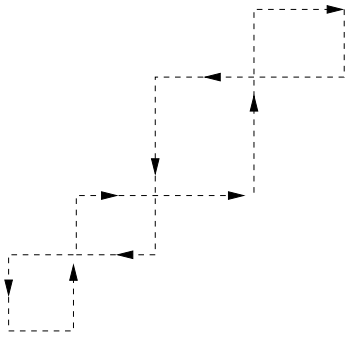

Abbildung 3.9: *Diese Endlosschleife mit drei Selbstschnitten erfüllt zwar die Voraussetzung, dass sich der Winkelzähler an einem bestimmten Punkt der Schleife über die Zeit nicht ändert. Allerdings kann solch eine Schleife nicht von einem Pledge-Algorithmus gefahren werden, da die parallelen Strecken nicht in die gleiche Richtung führen.*

Aufgrund der möglichen Endlosschleifen stellt sich die Frage, ob ein Algorithmus, der mit der Pledge-Steuerung arbeitet, überhaupt sinnvoll ist, um eine Labyrinth mit Einbahnstraßen zu lösen.

Lemma 3.8*: Der Pledge-Algorithmus kann ein Labyrinth mit Einbahnstraßen verlassen, vorausgesetzt es ist bekannt, ob eine Einbahnstraße durchlaufen werden soll, oder nicht.*

Beweis. Gegeben: ein faires Labyrinth L_{ein} mit Einbahnstraßen.
Seien alle Einbahnstraßen geschlossen, also als Wände zu betrachten.
Es existiert mindestens ein Gebiet G_1 von dem der Roboter das Labyrinth verlassen kann. Da alle Einbahnstraßen als Wände betrachtet werden, enthält das Gebiet G_1 keine Einbahnstraßen und einen Ausgang. Also findet ein Roboter mit einer Pledge-Steuerung im Gebiet G_1 immer den Ausgang.
Nun betrachtet man $L_{\text{ein}} \setminus \overline{G_1}$[2], wobei die Einbahnstraßen aus L_{ein} nach G_1 die zusätzlichen Ausgänge von $L_{\text{ein}} \setminus \overline{G_1}$ darstellen. Wie oben existiert wiederum mindestens ein Gebiet G_2 aus dem man $L_{\text{ein}} \setminus \overline{G_1}$ mit dem Pledge-Algorithmus verlassen kann.
Dies kann so lange durchgeführt werden, bis gilt: $L_{ein} \setminus \{\overline{G_1}, ..., \overline{G_n}\} = \emptyset$. Wenn nun der Roboter in das Gebiet G_m gesetzt wird, so gibt es eine Kette von Gebieten, die nach G_1 führt. Es werden die Einbahnstraßen, die nach G_{m-1} führen, auf „durchgehen“ gesetzt. Ebenso die Einbahnstraßen von G_{m-1} nach G_{m-2} usw. Alle übrigen

[2] $\overline{G_1}$ist der Abschluss des Gebietes G_1. Dies ist das Gebiet G_1 und alle angrenzenden Wände, die nicht mit einem anderen Gebiet geteilt werden.

Einbahnstraßen werden auf „geschlossen“ gesetzt.

■

Um mit einer Pledge-Steuerung den Ausgang des Labyrinthes zu erreichen, müssen also die Einbahnstraßen richtig konfiguriert werden. Sie müssen auf *geschlossen* oder *offen* gesetzt werden.

Definition 3.9*: Verhaltensvorschrift*
Eine Verhaltensvorschrift gibt an, ob der Roboter durch eine Einbahnstraße durchgehen oder nicht durchgehen soll.

Es gibt zwei Möglichkeiten eine Verhaltensvorschrift an Einbahnstraßen anzugeben.

- *Lokale Verhaltensvorschrift:* Eine lokale Verhaltensvorschrift ist eine Sequenz der Grundverhaltensweisen $\{$*durchgehen, nicht durchgehen*$\}$. Diese Vorschrift wird der Reihe nach abgearbeitet, ohne einen Zusammenhang mit bestimmten Einbahnstraßen. Ein Beispiel ist die Verhaltensvorschrift: „An jeder zweiten Einbahnstraße, die erreicht wird, durchgehen.“ Lokale Verhaltensweisen müssen nicht periodisch sein.

- *Globale Verhaltensvorschrift:* Eine globale Verhaltensvorschrift beschreibt eine Abbildung $f : M \longrightarrow \{$*geschlossen,offen*$\}$, wobei M die *Menge der Einbahnstraßen* ist. Eine globale Verhaltensvorschrift ist für jede Einbahnstraße eindeutig. Für die globale Verhaltensvorschrift ist es notwendig, dass die Einbahnstraßen unterscheidbar sind.

In Abbildung 3.7 ist ein Beispiel gegeben, in dem der Pledge-Algorithmus vom Startpunkt aus den Ausgang nicht mehr finden kann. Er gerät in eine Endlosschleife. Der Pledge-Algorithmus geht hier, wenn möglich, immer durch die Einbahnstraße hindurch. Allerdings führt die Annahme, nie durch eine Einbahnstraße zu gehen, ebenso wenig zum Ziel. Als Beispiel kann man sich ein Labyrinth vorstellen, dessen Ausgang eine Einbahnstraße ist.

Dies bedeutet, dass weder die Strategie *immer durchgehen*, noch die Strategie *nie durchgehen* zum Ziel führen.

Periodische, lokale Verhaltensvorschriften

Die einfachen lokalen Verhaltensweisen *immer durchgehen* und *nie durchgehen* führen nicht immer zum Ziel. Aber auch etwas kompliziertere Regeln wie z.B. *jedes zweite*

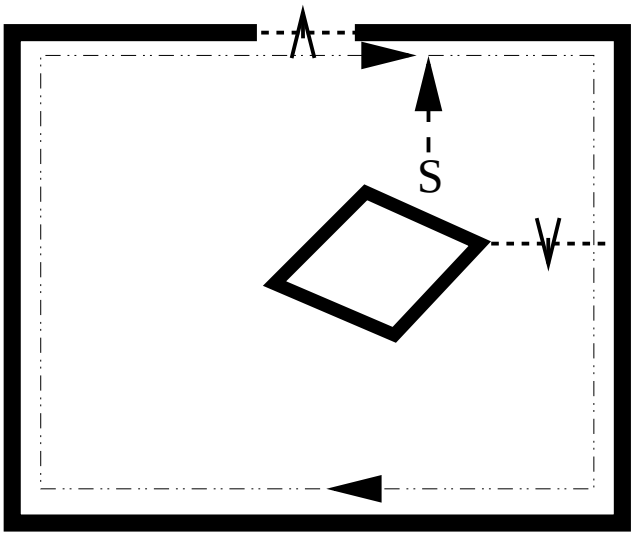

Abbildung 3.10: *Ein Gegenbeispiel für die Strategie:* Jedes zweite Mal nicht durchgehen.

Mal nicht durchgehen führt nicht zum Ziel wie das Gegenbeispiel in Abbildung 3.10 zeigt.
Mit Hilfe dieses einfachen Beispiels kann man sämtliche Strategien, die genau ein einziges Mal *nicht durchgehen* in einer Periode haben, widerlegen.
Auch Strategien wie: Wiederhole immer *durchgehen, nicht durchgehen, nicht durchgehen, durchgehen* führen nicht in jedem Fall zum Ausgang, wie Abbildung 3.11 zeigt.

Hieraus lässt sich zu jeder periodischen, lokalen Verhaltensweise ein Gegenbeispiel erzeugen.

Theorem 3.10*: Es existiert keine periodische, lokale Verhaltensweise, die jedes Labyrinth löst.*

Beweis. Sei der Ausgang im Gebiet G_1. Ferner existiert in diesem Gebiet G_1 ein weiteres Gebiet G_2 welches vollständig von G_1 umschlossen ist. Der Startpunkt S liegt in G_2.
Um ein Gegenbeispiel für die periodische, lokale Verhaltensvorschrift zu erzeugen, wird G_2 wie folgt aufgebaut:
Der Aufbau wird vom Startpunkt aus im Uhrzeigersinn betrachtet:

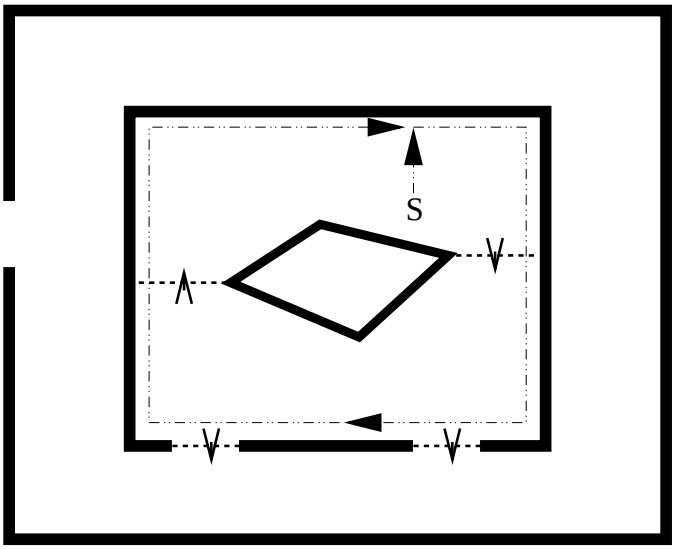

Abbildung 3.11: *Die periodische Verhaltensweise* durchgehen, nicht durchgehen, nicht durchgehen, durchgehen *führt nicht zum Ziel.*

- Wenn der nächste Schritt der Verhaltensweise *durchgehen* ist, so wird eine Einbahnstraße senkrecht zum Rand des Gebietes G_2 eingefügt (vgl. Abb. 3.11). Diese Einbahnstraße hat die durchfahrbare Seite so, dass der Roboter durchfahren kann. Sie trägt nicht zum Rand von G_2 bei.

- Wenn der nächste Schritt der Verhaltensweise *nicht durchgehen* ist, so wird ein Loch in den Rand des Gebietes G_2 eingefügt, welches mit einer Einbahnstraße mit Richtung von G_2 nach G_1 „geschlossen“ wird.

Dies wird für eine komplette Periode der lokalen Verhaltensweise durchgeführt. Das entstehende Labyrinth ist durch diese periodische, lokale Verhaltensweise nicht zu lösen. ■

Existenz von Verhaltensvorschriften

An diesen Beispielen ist zu sehen, dass es schwierig ist, **eine** lokale Verhaltensvorschrift zu finden, die **jedes** Labyrinth mit Einbahnstraßen löst. Allerdings gibt es für jedes Labyrinth sowohl mindestens eine globale Verhaltensvorschrift, als auch lokale Verhaltensvorschriften, die den Roboter zum Ausgang führen.

Lemma 3.11*: Es existiert zu jedem Labyrinth eine jeweilige globale Verhaltensvorschrift, die den Roboter zum Ziel führt.*

Beweis. Aus Lemma 3.8 ergibt sich diese globale Verhaltensvorschrift. Von jedem Gebiet G_m gibt es einen Weg durch andere Gebiete zu dem Gebiet G_1, welches den Ausgang enthält. Seien $G_{m-1}, G_{m-2} \ldots, G_2$ diese Kette von Gebieten die hierbei durchlaufen werden müssen[3]. Also müssen alle Einbahnstraßen, die von G_m nach G_{m-1} führen, auf *geöffnet* gesetzt werden. Alle Einbahnstraßen von G_{m-1} nach G_{m-2} ebenfalls, usw. Dies wird für alle Gebiete $G_i \;\; i = 1, \ldots, n$; $n =$ Anzahl Gebiete, durchgeführt. Alle anderen Einbahnstraßen werden *geschlossen*.
Hieraus ergibt sich eine globale Verhaltensvorschrift für ein Labyrinth, welche unabhängig von der Startposition des Roboters ist. ■

Lemma 3.12*: Es existiert zu jedem Startpunkt in einem Labyrinth mit Einbahnstraßen eine lokale Verhaltenvorschrift, die den Roboter zum Ausgang führt.*

Beweis. Aus Lemma 3.11 ist bekannt, dass es eine globale Verhaltensvorschrift für jedes Labyrinth gibt. Je nach Startpunkt kann diese globale Verhaltensvorschrift in eine lokale Verhaltensvorschrift umgesetzt werden. ■

Wie in den vorherigen Abschnitten gezeigt, muss der Pledge-Algorithmus so modifiziert werden, dass ihm die Möglichkeit gegeben wird, die richtige Verhaltensvorschrift herauszufinden. Es besteht die Möglichkeit eine „globale“ oder „lokale“ Verhaltensvorschrift zu suchen.

Da bei den Algorithmen häufig mit Bewegungen des Roboters und mit Auftreffpunkten auf bestimmte Teile eines Gebiets argumentiert werden, soll hier erst einmal eine Grundlage für die Verständigung geschaffen werden. Hierfür werden einige Definition benötigt:

Definition 3.13 *Rand eines Gebietes*
Der Rand eines Gebietes sind alle Wände, die das Gebiet mit anderen Gebieten teilt.

Definition 3.14 *Ausgang eines Gebietes*
Die Ausgänge des Gebietes G sind alle Einbahnstraßen, die von G in ein anderes Gebiet führen.

Definition 3.15 *Wichtiger Ausgänge eines Gebietes*
Jeder Ausgang E aus dem Gebiet G ist genau dann ein wichtiger Ausgang, wenn es vom Startpunkt aus einen Weg zu Ausgang des Labyrinthes gibt, so dass E überfahren werden muss.

[3] diese Kette muss nicht eindeutig sein

Definition 3.16 *Innere Außenrand eines Gebietes Zum inneren Außenrand des Gebietes gehören alle die Wände und Einbahnstraßen, die berührt werden, wenn der Roboter von einem wichtigen Ausgang aus, mit Hilfe der linken Hand Regel an der Wand langgeht, ohne dabei Einbahnstraßen zu durchlaufen.*

Definition 3.17 *Außenrand eines Gebietes*
Der Außenrand eines Gebietes besteht aus den Wände und Einbahnstraßen des inneren Außenrandes, die zu zwei unterschiedlichen Gebieten gehören.

Korollar 3.18*: Wichtige Ausgänge liegen auf dem Außenrand*

Beweis. Ausgänge, die zwar auf dem Rand des Gebietes G aber nicht auf dem Außenrand liegen, führen in Gebiete, die komplett von G eingeschlossen sind. So muß nicht durch diese Ausgänge gefahren werden, um den Ausgang des Labyrinths zu erreichen. ■

3.2 Einbahnstraßen-Pledge Typ1

Mit der Bezeichnung Einbahnstraßen-Pledge Typ1 werden im folgenden Algorithmen bezeichnet, die den Roboter mit Hilfe eines Pledge-Algorithmus durch ein Labyrinth mit Einbahnstraßen steuern. Da an den Einbahnstraßen eine Entscheidung getroffen werden muss, ist zusätzlich zu dem Pledge-Algorithmus noch eine Kontrollstruktur notwendig, die dem Algorithmus ermöglicht, diese Entscheidungen zu treffen. In den Algorithmen von Typ1 sind die Einbahnstraßen für den Roboter nicht unterscheidbar. Der Roboter muss also eine **lokale Verhaltensweise** finden, die ihn aus dem Labyrinth führt.
Im Rahmen dieser Diplomarbeit ist ein Einbahnstraßen-Pledge Typ1 Algorithmus entwickelt worden. Dieser Algorithmus findet eine lokale Verhaltensweise mit der Hilfe eines Steuerwortes.

3.2.1 Steuerwort-Pledge

Der Steuerwort Einbahnstraßen-Pledge Typ1, kurz Steuerwort-Pledge wird mit Hilfe eines Steuerworts realisiert. Die Grundsteuerung des Roboters ist ein Pledge-Algorithmus. Das Steuerwort gibt dem Algorithmus an, welche Aktion an einer Einbahnstraße auszuführen ist. Wenn der Algorithmus durch die Pledge-Steuerung an eine Einbahnstraße gelangt und durchfahren will, wird der nächste Buchstabe des Steuerwortes ausgewertet und dementsprechend verfahren. Sobald der Roboter das Signal bekommt, dass der Ausgang erreicht ist, wird der Algorithmus abgebrochen. Dies bedeutet, dass das Alphabet Σ aus mindestens dem folgenden Zeichenvorrat bestehen muss:

- „d“ entspricht *durchgehen*

- „n“ entspricht *nicht durchgehen*

Jedes endliche Wort das aus diesen Buchstaben gebildet werden kann, entspricht einer nicht-periodischen lokalen Verhaltensweise. Zusätzlich wird noch ein drittes Zeichen eingeführt:

- „r“ entspricht *Winkelzähler auf Null setzen*

Das Zeichen „r“ führt an einer Einbahnstraße nicht weiter. Es ist nur dazu gedacht, den Pledge-Algorithmus an dieser Einbahnstraße neu zu starten. Damit der Roboter sich von dieser Einbahnstraße wegbewegt, muss natürlich mindestens ein „d“ oder ein „n“ folgen.

Theorem 3.19*: Es existiert für jedes Labyrinth ein endliches, universelles Steuerwort w_{uni} über Σ, das den Roboter zum Ausgang führt.*

Beweis. Sei A die erste Einbahnstraße auf die der Roboter trifft. So ist aus Lemma 3.8 bekannt, dass es eine lokale Verhaltensvorschrift gibt, die den Roboter von A aus zum Ausgang führt. Sei diese Verhaltensvorschrift das Wort w_A. Dieses Wort ist endlich. Als nächstes treffe der Roboter vom Startpunkt auf eine weitere Einbahnstraße $B \neq A$. Es wird w_A ausgeführt. Entweder der Roboter findet hierdurch schon den Ausgang, oder der Roboter stoppt an der nächsten Einbahnstraße, nachdem das Wort w_A ausgeführt wurde. Wenn von hier aus der Pledge-Algorithmus neu gestartet wird, gibt es von dieser Einbahnstraße das Wort w_B, welches dann zum Ziel führt und ebenfalls endlich ist. Also führt das Wort $w_{AB} = w_A \oplus (r) \oplus w_B$ schon von zwei Einbahnstraßen zum Ausgang.
Dies kann für alle Einbahnstraßen weitergeführt werden, so dass zum Schluss mit Hilfe des resultierenden Wortes w_{uni} von allen Einbahnstraßen aus der Pledge-Algorithmus das Labyrinth verlassen kann. Dieses Wort hat eine endliche Länge, da alle Unterworte ebenfalls endliche Länge haben und nur endlich oft der Buchstabe „r“ eingefügt wird. ■

Dieses Wort w_{uni} ist für verschiedene Labyrinthe unterschiedlich und dem Roboter natürlich nicht bekannt.
Der Steuerwort-Pledge geht zum lösen den Labyrinth wie folgt vor. Der Roboter führt jedes Steuerwort der Länge eins aus. Wenn dies nicht zum Ziel führt, dann probiert er jedes Steuerwort der Länge zwei. Dies wird maximal so weit geführt, bis das universelle Steuerwort w_{uni} für dieses Labyrinth gefunden wird. Da es von jeder Einbahnstraße aus den Ausgang findet, führt dieses Verfahren mit Sicherheit zum Ausgang.

Dieses Verfahren zeigt, dass es immer möglich ist, mit Hilfe des Pledge-Algorithmus und eines Steuerwortes ein Labyrinth mit Einbahnstraßen zu verlassen. Um die Länge von w_{uni} abzuschätzen wird die Länge eines Steuerwortes von einer Einbahnstraße zum Ausgang betrachtet:

Lemma 3.20*: Ein Steuerwort, welches mit Hilfe des Steuerwort-Pledge Typ1 von einer beliebigen Einbahnstraße zum Ausgang führt, liegt in $O(N)$, wobei N die Anzahl der Einbahnstraßen ist.*

Beweis. Der Weg zum Ausgang lässt sich als Kette von Gebieten beschreiben. Der Roboter startet in Gebiet G_m und muss in das Gebiet G_1, da dort der Ausgang liegt. So muss er nacheinander die Gebiete $G_{m-1}, G_{m-2} \dots G_2$ durchfahren. Wenn der Roboter nun an eine Einbahnstraße kommt, so führt diese Einbahnstraße von genau einem Gebiet in genau ein Gebiet[4]. Führt diese Einbahnstraße nun den Roboter in der Kette weiter, so wird die Einbahnstraße durchfahren. Wenn dieser Übergang nicht in der Kette ist, wird sie nicht durchfahren. Jede Einbahnstraße wird dabei nur einmal betrachtet, da der Roboter spätestens nach einem Umlauf im Gebiet alle Einbahnstraßen gefunden hat, die aus dem Gebiet hinausführen und eine davon durchfahren hat. Dies bedeutet, es gibt zusammen maximal N mal die Buchstaben 'd' und 'n' in diesem Steuerwort. Sinnvoll kann ein 'r' nur einmal eingefügt werden. Zwei 'r' hintereinander haben die gleiche Auswirkung, wie ein einzelnes 'r'. Dies bedeutet zum Steuerwort kommen noch maximal N 'r' hinzu. Das Steuerwort besteht also aus maximal $2N$ Buchstaben und liegt somit in $O(N)$. ■

Das universelle Steuerwort ist eine Verkettung von N Steuerworten der Länge $2N$. Das universelle Steuerwort hat demnach eine Länge von $2N^2$ Buchstaben.
Dies führt zu dem Theorem:

Theorem 3.21*: Es müssen maximal $3^{O(N^2)}$ Wörter untersucht werden, um w_{uni} zu finden.*

Beweis. Nach Lemma 3.19 existiert ein universelles Steuerwort und nach Lemma 2.12 hat es maximal die Länge $2 \cdot N^2$.
Dies bedeutet, dass der Roboter spätestens nachdem er alle Steuerworte der Länge $2 \cdot N^2$ ausprobiert hat, das Ziel erreicht. Die Anzahl der Wörter, die bis dahin ausprobiert wurden, ist:

$$\Sigma_{i=1}^{2N^2} 3^i \leq N^2 \cdot 2^{n^2} = 3^{O(n^2)}$$

■

Aufgrund dieser Laufzeit, ist der Steuerwort-Pledge Typ1 schon bei wenigen Einbahnstraßen nicht mehr auf einem realen Roboter praktikabel. Deswegen wird nach einem Algorithmus gesucht, der auf einem realen Roboter genutzt werden kann. Hierfür wird nach einer globalen Verhaltensweise gesucht.

[4]Einbahnstraßen, die nicht zu einem Gebiet beitragen, führen von einem Gebiet in das Selbe

3.3 Einbahnstraßen-Pledge Typ2

Die Bezeichnung Einbahnstraßen-Pledge Typ2 ist, analog zum Einbahnstraßen-Pledge Typ1 eine Sammelbezeichnung für Algorithmen, die ein Labyrinth mit Einbahnstraßen lösen. Im Gegensatz zu Algorithmen des Einbahnstraßen-Pledge Typ1, haben diese jedoch ein Roboter Modell, welches die Fähigkeit besitzt, die Einbahnstraßen voneinander unterscheiden zu können. Dies bedeutet einen erheblichen Informationsvorteil gegenüber Einbahnstraßen-Pledge Typ1 Algorithmen.
Es wurden im Rahmen dieser Diplomarbeit drei verschiede Algorithmen vom Typ2 entwickelt. Jeder dieser Algorithmen nutzt die Information, die durch die Identifizierbarkeit der Einbahnstraßen ergeben, anders. Gemeinsam ist alle Algorithmen, dass sie einen Speicherbedarf von $O(N)$ haben, wobei N die Anzahl der Einbahnstraßen ist.

3.3.1 Binärzahl-Pledge

Der Binärzahl-Pledge sucht nach einer globalen Verhaltensweise, in dem er alle möglichen Verhaltensweisen ausprobiert. Die Informationen, die durch die Identifizierung der Einbahnstraßen gewonnen werden, werden hier genutzt, um zu erkennen, wenn eine Verhaltensweise nicht zum Ziel führt.

Beschreibung des Algorithmus

Es ist bekannt, dass es mindestens eine globale Verhaltensweise gibt, die den Roboter zum Ziel führt. Wichtig hierbei ist, dass diese globale Verhaltensliste unabhängig vom Startpunkt des Roboters ist. Angenommen, dem Roboter wäre es bekannt, wie viele Einbahnstraße das Labyrinth enthält. Angenommen dies seien N Einbahnstraßen. Dann ließe sich eine Verhaltensweise durch eine N-stellige Binärzahl darstellen. Hierbei wird jeder Stelle der Binärzahl eine bestimmte Einbahnstraße zugeordnet. Ein 0 bedeutete, dass diese Einbahnstraße nicht durchfahren werden soll und eine 1, dass die Einbahnstraße durchfahren werden soll. Die Suche nach einer globalen Verhaltensweise, die den Roboter zum Ziel führt, bedeutet, alle N-stelligen Binärzahlen auszuprobieren, so lange, bis der Roboter das Labyrinth verlassen hat.
Die Schwierigkeit hierbei ist zu entscheiden, wann eine Verhaltensweise mit Sicherheit nicht zum Ausgang führt. Nur dann darf die nächste Verhaltensweise versucht werden. Mit Hilfe der unterscheidbaren Einbahnstraßen hat der Roboter im Labyrinth Landmarken, die er zur Navigation benutzen kann. Er kann also erkennen, wenn er im Kreis fährt. Um dies zu ermöglichen, wird eine Datenstruktur gebraucht, die speichert, von wo nach wo der Roboter mit der aktuellen Verhaltensweise gefahren ist. Für jede Einbahnstraße wird nicht nur abgespeichert, ob der Roboter durchfahren soll, oder nicht[5],

[5] Mit Hilfe der Binärzahl

sondern auch mit welchem Winkelzähler er die letzten Male vorbeigekommen ist. Eine besondere Bewegung des Roboters ist die „freie Bewegung“. Der Roboter speichert eine Flag, sobald er eine freie Bewegung macht.
Wird nun ein Kreis gefunden, d.h. der Roboter fährt zweimal exakt die gleichen Einbahnstraßen ab, so kann man sich die Veränderungen der Winkelzählerwerte an diesen Einbahnstraße anschauen. Es ergeben sich folgende Beziehungen:

	keine freie Bewegung	freie Bewegung
mindestens einer der Winkelzählerwerte ist größer geworden		
alle Winkelzählerwerte sind gleich geblieben		
alle Winkelzählerwerte sind kleiner geworden		
kein Winkelzählerwert ist größer geworden, aber mindestens einer kleiner und mindestens einer gleich		

Hierbei wird die Veränderung an jeweils einer Einbahnstraße bei Durchlauf eins und zwei betrachtet. So bedeutet z.B. „alle Winkelzähler sind kleiner geworden“, dass an jeder Einbahnstraße im Kreis, der Winkelzähler des Roboters beim zweiten Besuch kleiner war, als beim ersten. Die Aussage „mindestens einer der Winkelzählerwerte ist größer geworden“ bedeutet, dass es mindestens eine Einbahnstraße im Kreis gibt, an der der Winkelzähler des Roboters beim zweiten Besuch im Vergleich zum ersten Besuch größer geworden ist.
Da dies alle Möglichkeiten der beiden Parameter (Winkelzähler-Entwicklung und freie Bewegung) sind, wurde untersucht, ob diese aussagekräftig dafür sind, ob der Roboter sich in einer Endlosschleife befindet. Befindet er sich in solch einer, so ist gezeigt, dass die aktuelle globale Verhaltensweise nicht zum Ziel führt. Also kann die nächste Binärzahl ausprobiert werden.
Im folgenden werden die verschiedenen Varianten untersucht. Es ist dabei zu beachte, dass zu dem Zeitpunkt ein Kreis entstanden ist, der zweimal durchlaufen wurde. Während den ganzen Betrachtungen wird die Verhaltensweise nicht geändert. Es ist zu erwähnen, dass es ausreicht, die Winkelzählerwerte an den Einbahnstraßen zu betrachten. Die Tendenzen des Winkelzählers der Einbahnstraßen werden zwischen diese fortgesetzt.

- Es existiert mindestens eine freie Bewegung im festgestellten Kreis
  - Mindestens einer der Winkelzählerwerte ist größer geworden: Da der Winkelzähler nicht größer als Null werden kann, wird sich hier, bei mehrfach erneutem Durchlaufen etwas ändern.
  - Alle Winkelzählerwerte sind gleich geblieben: Die Verhaltensweise bleibt gleich. Deswegen wird der Roboter immer wieder genau diese Route fahren. Er ist in einer Endlosschleife.
  - Alle Winkelzählerwerte sind kleiner geworden: Da der Roboter mindestens eine freie Bewegung ausgeführt hat, ist an mindestens einer Stelle der Winkelzähler auf Null gekommen. Wird der Kreis immer weiter gefahren und alle Winkelzählerwerte immer kleiner, wird sich der Weg ändern, sobald der Winkelzähler klein genug ist, dass die freie Bewegung nicht mehr möglich ist.
  - Kein Winkelzählerwert ist größer geworden, aber mindestens einer kleiner und einer gleich: Da der Roboter mindestens eine freie Bewegung macht und sich mindestens einer der Winkelzählerwerte kleiner wird, wird die freie Bewegung irgendwann nicht mehr möglich sein. Der Weg wird sich also ändern, wenn dieser Punkt erreicht ist.
- Es existiert keine freie Bewegung im festgestellten Kreis
  - Mindestens einer der Winkelzählerwerte ist größer geworden: Da der Winkelzähler nicht größer als Null werden kann, wird sich hier, bei mehrfach erneutem Durchlaufen etwas ändern.
  - Alle Winkelzählerwerte sind gleich geblieben: Die Verhaltensweise bleibt gleich. Deswegen wird der Roboter immer wieder genau diese Route fahren. Er ist in einer Endlosschleife.
  - Alle Winkelzählerwerte sind kleiner geworden: Da der Roboter keine freie Bewegung mehr ausführt, wird der Weg immer der gleiche bleiben.
  - Kein Winkelzählerwert ist größer geworden, aber mindestens einer kleiner und einer gleich: Da keine freie Bewegung vom Roboter ausgeführt wird und eine solche freie Bewegung nicht mehr möglich wird (da kein Winkelzählerwert größer wird), ändert sich der Weg des Roboters nicht mehr

Mit dieser Betrachtung ergibt sich folgende Verhältnisse:

	keine freie Bewegung	freie Bewegung
mindestens einer der Winkelzählerwerte ist größer geworden	Weg ändert sich	Weg ändert sich
alle Winkelzählerwerte sind gleich geblieben	Weg ändert sich **nicht**	Weg ändert sich **nicht**
alle Winkelzählerwerte sind kleiner geworden	Weg ändert sich **nicht**	Weg ändert sich
kein Winkelzählerwert ist größer geworden, aber mindestens einer kleiner und mindestens einer gleich	Weg ändert sich **nicht**	Weg ändert sich

Mit Hilfe dieser Beziehungen sind die Bedingungen definiert, an denen die Binärzahl geändert wird. Bis hier her wurde angenommen, dass dem Roboter bekannt ist, wie viele Einbahnstraßen existieren. Beim Starten des Algorithmus ist dem Roboter allerdings noch keine Einbahnstraße bekannt.
Der Roboter beginnt mit der Pledge-Steuerung. Trifft er auf keine Einbahnstraße und findet nicht heraus, so gibt es keine Ausgang (vgl. Korrektheit des Pledge Algorithmus). Trifft er hingegen auf Einbahnstraßen, wird gemäß der ersten Binärzahl 0 keine Einbahnstraße durchfahren. Die bekannte Umgebung, mit den bekannten Einbahnstraßen, kann als Sub-Labyrinth betrachtet werden. Für diese Sub-Labyrinth gibt es auch eine Verhaltensweise, die der Roboter findet.
Der Algorithmus startet mit einer leeren Liste, in der die Einbahnstraßen gespeichert werden. Dementsprechend ist auch die Binärzahl zur Steuerung an den Einbahnstraßen leer. Das Flag für eine freie Bewegung wird auf Null gesetzt.
Der Algorithmus läuft wie folgt:
Der Roboter wird durch den Pledge-Algorithmus gesteuert. Wird hierbei eine freie Bewegung registriert, so wird das „freie Bewegung-Flag“ gesetzt. Trifft der Roboter auf eine Einbahnstraße, wird geschaut ob diese in der Liste ist. Dementsprechend wird weiter verfahren:

- Die Einbahnstraße ist nicht vorhanden. So wird in die Liste eingefügt. Alle Verbindungen von einer Einbahnstraße zur anderen werden gelöscht. Die Binärzahl wird auf $N \cdot 0$ gesetzt, wobei N die Anzahl der bekannten Einbahnstraßen ist. Das „freie Bewegung-Flag“ wird auf Null gesetzt. Alle Einbahnstraßen werden als „nicht besucht“ markiert. Die neu gefundene Einbahnstraße wird als „besucht“ markiert. Der aktuelle Winkelzählerwert wird gespeichert. Der Roboter fährt weiter mit dem Pledge Algorithmus. Die gefundene Einbahnstraße wird nicht durchfahren.

- Die Einbahnstraße ist vorhanden. Es wird eine Verbindung von der letzten besuchten Einbahnstraße zu der jetzigen gespeichert. Der aktuelle Winkelzählerwert wird gespeichert. Die Liste der Einbahnstraßen wird auf einen Kreis untersucht. Hierbei können zwei Dinge passieren:
  - Es existiert kein Kreis, der schon zweimal durchlaufen wurde. Der Roboter fährt weiter mit dem Pledge Algorithmus. Die gefunden Einbahnstraße wird gemäß Binärwort behandelt.
  - Es existiert ein Kreis, der schon zweimal durchlaufen wurde. Nun wird nach den oben genannten Kriterien entscheiden ob die Binärzahl inkrementiert werden soll.
    * Die Binärzahl wird nicht hochgezählt. Der Roboter wird mit Pledge-Algorithmus weitergeführt. Die Einbahnstraßen werden auf geschlossen oder offen gesetzt, so wie die Binärzahl es vorgibt. Das „freie Bewegung-Flag“ wird auf Null gesetzt. Die gespeicherten Winkelzählerwerte für die Einbahnstraßen werden gelöscht.
    * Die Binärzahl wird um eins hochgezählt. Alle Verbindungen zwischen den Einbahnstraßen werden gelöscht. Das „freie Bewegung-Flag“ wird auf Null gesetzt. Es werden alle gespeicherten Winkelzählerwerte gelöscht. Der Winkelzähler wird auf Null gesetzt. Dann wird der Roboter mit Pledge-Algorithmus weitergeführt. Die gefunden Einbahnstraße wird gemäß Binärwort behandelt.

Der Speicherbedarf des Algorithmus hängt von der Datenstruktur ab, in der die Einbahnstraßen gespeichert werden und von der Komplexität, den ein Kreis haben kann. Kann in einem Kreis jede Einbahnstraße nur einmal vorkommen, so ist der Speicherbedarf des Algorithmus in $O(N)$, wobei N die Anzahl der Einbahnstraßen sind.
Ein Kreis zu detektieren ist nur dann wichtig, wenn dieser eine Endlosschleife darstellt. Um die Komplexität der zu detektierenden Kreise zu betrachten, müssen also nur Kreise betrachtet werden, die die Kriterien für Endlosschleifen erfüllen.
Angenommen, es existiert ein Kreis, in dem Einbahnstraßen mehrfach besucht werden. Dies bedeutet, dass nach einer Einbahnstraße, die mehrfach in diesem Kreis vorkommt, unterschiedliche Wege folgen. Damit dies geschehen kann, muss dieser Kreis einen Selbstschnitt haben. Dies bedeutet, der Kreis enthält mindestens eine freie Bewegung (siehe Lemma 3.6). Dies bedeutet, solch ein Kreis ist für den Algorithmus nur wichtig, wenn der Winkelzähler sich nicht an jeder Einbahnstraße nach jedem vollen Umlauf ändert. Dies bedeutet, der Kreis hat genau einen Selbstschnitt. Die Einbahnstraße, die mehrfach (nämlich zwei mal) im Kreis gefunden wurde, ist die Einbahnstraße, die den Selbstschnitt auslöst. Dies bedeutet aber, dass die freie Bewegung durch die Einbahnstraße geht (sonst hätte der Roboter die Einbahnstraße nicht besucht) und die „nicht freie“ Bewegung an der Einbahnstraße entlang. Da sich die Verhaltensweise während

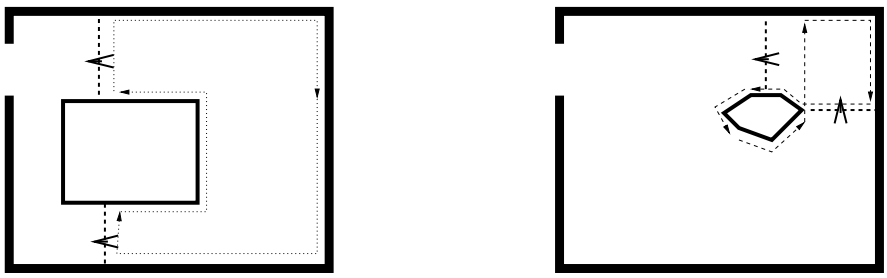

Abbildung 3.12: *Zwei Beispiele für einen Weg des Roboter, bei dem der Binärzahlen-Pledge die Binärzahl ändern muss. Im linken Bild sind beide Einbahnstraße auf geschlossen gestellt. So werden die Winkelzählerwerte immer kleiner, da der Roboter in einem Innehof fährt. Im rechten Labyrinth sind beide Einbahnstraße, die der Roboter besucht geöffnet. Hierduch entsteht ein Weg der einen Selbstschnitt hat. Die Winkelzählerwerten an den Einbahnstraßen ändern sich mit der Zeit nicht*

des Kreises nicht ändert, muss also die „nicht freie“ Bewegung an der verbotenen Seite der Einbahnstraße geschehen. Dies bedeutet aber, dass der Roboter die Einbahnstraße nicht gehen hat, sondern als Wand wahrgenommen hat. Daraus folgt, dass es keinen relevanten, weil unendlichen Kreis gibt in dem eine Einbahnstraße mehr als einmal besucht wird.
Der Speicherbedarf des Binärzahl-Algorithmus liegt also in $O(N)$.

Korrektheit des Algorithmus

Um die Korrektheit des Binärzahlen-Pledge zu beweisen, müssen zuerst die Bedingungen überprüft werden, bei denen die Binärzahl gewechselt wird. Aus den Winkelzählerwerten und der Angabe, ob eine freie Bewegung gemacht wird oder nicht, folgt, ob der Roboter sich in einer Endlosschleife aufhält oder nicht. Für die Betrachtung der Bedingungen gilt immer, dass sich während der Fahrt des Roboters die Verhaltensweise nicht ändert. Zusätzlich beschreibt der Weg eine geschlossene Kurve, die in dieser Form schon zwei mal durchlaufen wurde, wie anhand der Einbahnstraßen festgestellt wurde. Nur dann wird auf die Bedingungen überprüft. Mit Winkelzählerwerten sind immer die abgespeicherten Winkelzählerwerte der Einbahnstraßen gemeint, die innerhalb des Kreises besucht werden.

Korollar 3.22*: Wenn alle Winkelzählerwerte bei zwei Umläufen gleich bleiben, so ist der Roboter in einer Endlosschleife gefangen, ungeachtet ob eine freie Bewegung in dieser Schleife ist oder nicht.*

Beweis. Da die Steuerung durch den Algorithmus deterministisch ist und die Verhaltensweise an den einzelnen Einbahnstraßen nicht geändert wird, ändert sich an dem Weg des Roboters nichts mehr, wenn die Winkelzählerwerte alle gleich bleiben. Zwischen zwei Einbahnstraßen kann sich dann auch nichts mehr ändern, da dazwischen der Roboter durch den Pledge-Algorithmus gesteuert wird. Eventuelle Absprungpunkte bleiben dementsprechend gleich. So ergibt sich trotz eventueller freier Bewegung eine endlose Schleife die immer wieder durchlaufen wird. ■

Mit diesem Korollar sind beide Bedingungen der gleich bleibenden Winkelzählerwerte abgedeckt. Es fehlen noch zwei Bedingungen die laut Algorithmus zu einer Endlosschleife führen.

Korollar 3.23*: Wenn alle Winkelzählerwerte bei zwei Umläufen kleiner werden und es keine freie Bewegung gibt, so ist der Roboter in einer Endlosschleife.*

Beweis. Bei Winkelzählerwerten kleiner Null führt der Roboter eine Wall-Follower Bewegung aus. Da ein Kreis entdeckt wurde vollführt der Roboter eine Bewegung immer an der Wand entlang. Um aus diesem Kreis auszubrechen müsste der Roboter eine freie Bewegung ausführen. Da die Winkelzählerwerte allerdings immer kleiner werden, kann keine freie Bewegung mehr auftreten. ■

Korollar 3.24*: Wenn kein Winkelzählerwert größer wird, die anderen aber sowohl kleiner, oder gleich bleiben und der Roboter keine freie Bewegung macht, so ist der Roboter ebenfalls im Wall-Follower Modus.*

Beweis. Da ein Kreis gefunden wurde, könnte der Roboter sich nur aus diesem Kreis lösen, wenn er eine freie Bewegung durchführt. Dies ist nicht möglich, da zu Beginn des Kreises keine freie Bewegung durchgeführt wurde und bei mehrmaligen Durchlaufen kein Winkelzählerwert größer wird. ■

Hiermit ist gezeigt, dass die Bedingungen für eine Endlosschleife diese auch wirklich repräsentieren. Nun müssen noch die restlichen Bedingungen überprüft werden, ob sie sich tatsächlich bei mehrmaligen Durchlaufen ändern.

Korollar 3.25*: Wenn mindestens einer der Winkelzählerwerte größer wird, ist der Roboter nicht in einer Endlosschleife.*

Beweis. Da der Roboter eine geschlossene Kurve läuft, wird der Winkelzähler immer größer. Er kann nicht größer Null werden, da der Roboter sich vorher von der Wand löst. Also wird dieser Kreis durchbrochen. ■

Korollar 3.26*: Wenn alle Winkelzählerwerte kleiner werden, der Roboter allerdings noch eine freie Bewegung macht, so ist dies keine Endlosschleife.*

Beweis. Wenn der Roboter eine freie Bewegung macht, so bedeutet es, an mindestens einem Punkt des geschlossenen Weges ist der Winkelzähler gleich Null. Werden alle Winkelzählerwert nun immer kleiner, so kann an diesem Punkt keine freie Bewegung mehr stattfinden, da sonst das Labyrinth selber sich geändert haben müsste. Der Roboter hätte beim zweiten Umlauf auf der gleichen Strecke eine größere positive Änderung des Winkelzählers erhalten müssen. ■

Korollar 3.27*: Wenn kein Winkelzähler größer geworden ist, die anderen jeweils kleiner oder gleich geblieben sind und der Roboter eine freie Bewegung gemacht hat, so ist dies keine Endlosschleife.*

Beweis. Es gilt die gleiche Argumentation, wie für Korollar 3.26. ■

Nun kann die Korrektheit des Binärzahlen-Einbahnstraßen-Pledge gezeigt werden.

Theorem 3.28*: Der Binärzahlen-Einbahnstraßen-Pledge findet immer aus einem Labyrinth mit Einbahnstraßen.*

Beweis. Der Binärzahlen-Pledge erkennt, wie gezeigt, Endlosschleifen und wechselt daraufhin die Verhaltensweise. Er probiert alle globalen Verhaltensweisen. Da es nach Lemma 3.11 eine gültige Verhaltensweise gibt, findet der Roboter aus dem Labyrinth heraus. ■

3.3.2 Gebiets-Pledge

Der Gebiets-Pledge, nutzt die Informationen, die durch die identifizierbaren Einbahnstraßen gegeben sind am stärksten aus. Es ist bekannt, dass es eine Kette von Gebieten gibt, die vom Startpunkt des Roboters zum Ausgang führt. Diese Kette von Gebieten findet der Gebiets-Pledge. Im Gegensatz zu den anderen Einbahnstraßen-Pledge Typ2 Algorithmen wird der Roboter nicht ausschließlich mit dem Pledge-Algorithmus gesteuert.

Beschreibung des Algorithmus

Die Idee hinter dem Gebiets-Pledge ist die, dass erkannt werden soll, ob eine Einbahnstraße ein wichtiger Ausgang sein kann, oder nicht. Zusätzlich soll herausgefunden werden, in welches Gebiet sie führt, wenn sie ein wichtiger Ausgang ist.
Zu diesem Zweck wird der Roboter mit Hilfe des Pledge-Algorithmus gesteuert, bis er auf eine Einbahnstraße E_x stößt, welche nicht „fest geschlossen“[6] ist. Um herauszufinden, ob diese Einbahnstraße ein wichtiger Ausgang des aktuellen Gebietes ist, wird

[6]Das Label „fest geschlossen“ bedeutet, dass der ROboter niemals durch diese Einbahnstraße fahren muss. Es ist als abgrenzung zu dem Label „geschlossen“ zu sehen, da es hier durchaus sein kann, dass der Roboter später durch diese Einbahnstraße gehen muss, sie also noch geöffnet werden muss.

nun die Pledge-Steuerung abgeschaltet. Der Roboter fährt nun so, dass er die Wand immer an der rechten Seite hat. Er folgt der Wand so lange, bis er wieder an E_x stößt. Beim Verfolgen der Wand, kann der Roboter auf weitere Einbahnstraßen treffen. Diese werden in einer Liste gespeichert, aber nicht durchfahren. Wenn der Roboter wieder an E_x angekommen ist, entscheidet sich das weitere vorgehen des Algorithmus nach der Änderung des Winkelzählers, welcher weiterhin mitprotokolliert wird:

- Der Winkelzähler hat sich um 2π geändert. Der Roboter ist außen um ein inneres Hindernis gelaufen. Alle, auf diesem Weg gefundenen Einbahnstraßen sind keine wichtigen Ausgänge. Sie müssen nie durchlaufen werden. In der Liste, in der alle Einbahnstraßen gespeichert werden, werden diese Einbahnstraßen „fest geschlossen“. Es gibt keinen Grund diese jemals zu durchfahren. Der Roboter wird mit dem Pledge-Algorithmus neu gestartet, bis er an eine Einbahnstraße kommt, die er noch nicht getroffen hat.

- Der Winkelzähler hat sich um -2π geändert. Der Roboter ist am inneren Außenrand eines Gebietes entlang gefahren. Die gefundenen Einbahnstraßen sind potentielle wichtige Ausgänge. Der Roboter fährt durch den ersten dieser Ausgänge. Der Roboter wird nun mit dem Pledge Algorithmus weitergeführt. Ob die durchfahrene Einbahnstraße wirklich ein wichtiger Ausgang war, wird überprüft, sobald die nächste Einbahnstraße E_y getroffen wird. Hierbei gibt es zwei Möglichkeiten:

  - Die Einbahnstraße E_y gehört zu dem Gebiet, aus dem der Roboter gerade kommt. Dann war die Einbahnstraße E_x kein wichtiger Ausgang. Sie wird auf „fest geschlossen“ gesetzt. Der Roboter fährt weiterhin an der Wand entlang, bis er zur nächsten nicht „fest geschlossenen“ Einbahnstraße kommt und diese durchfährt um zu testen, ob sie ein wichtiger Ausgang ist.
  - Die Einbahnstraße E_y gehört zu einem anderen Gebiet. Dies kann auf zwei Arten erkannt werden:
    * Die Einbahnstraße E_y besitzt bereits ein Kennzeichnung eines anderen Gebietes.
    * Die Einbahnstraße E_y besitzt noch keine Kennzeichnung. Hier wird zuerst angenommen, dass sie zu einem anderen Gebiet gehört. Der Roboter verfolgt nun die Wand an der E_y liegt. Trifft er dabei auf keine bekannte Einbahnstraße und der Winkelzähler wird um -2π erhöht, so ist E_y in einem neuen Gebiet. Wird eine bekannte Einbahnstraße getroffen und sie gehört zu einem anderen Gebiet, so ist dem Roboter nun auch bekannt zu welchem Gebiet E_y gehört.

Dann kann in der Liste der Einbahnstraßen vermerkt werden, von welchem Gebiet in welches Gebiet die durchfahrene Einbahnstraße führt. Nun wird an der neuen Einbahnstraße wieder an der Wand entlang gefahren um zu testen, ob die neue Einbahnstraße ein wichtiger Ausgang darstellt.

Dies wird so lange weiter geführt, bis der Ausgang gefunden wurde. Während der Roboter durch das Labyrinth fährt und testet welche Einbahnstraßen wichtig sind, wird in der Liste eine Art Navigationshilfe immer weiter aufgebaut. Es entsteht dadurch ein Graph, dessen Knoten die einzelnen Gebiete sind und dessen Kanten die Einbahnstraßen sind, welche aus einem Gebiet ist das andere führt. Diese Kanten sind also gerichtet (für ein Beispiel siehe Abb. 3.13).

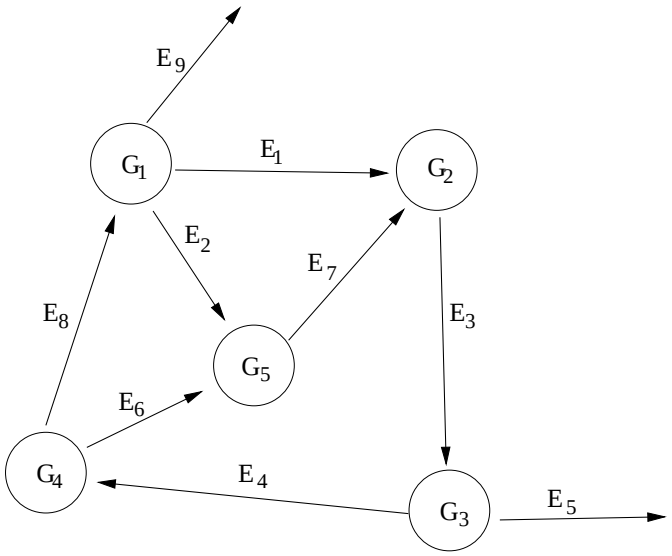

Abbildung 3.13: *Ein Bespielgraph des Gebiets-Pledge. Das Labyrinth ist noch nicht vollständig erkundet. Wohin die Einbahnstraßen E_5 und E_9 führen ist noch nihct bekannt. Mit Hilfe dieses Graphen kann der Roboter nun aber zu den Gebieten G_1 oder G_3 gesteuert werden, um die unbekannten Einbahnstraßen zu erkunden. Innere Hindernisse werden in solch einem Graphen nicht mher berücksichtigt, da sie für die Wegfindung irrelevant sind. Welche der unbekannten Einbahnstraßen als nächstes untersucht wird, kann über verschiedene Verfahren bestimmt werden. So z.B. welche dem Roboter am nächsten ist. Oder die Einbahnstraße, mit der gringsten Indexnummer.*

Korrektheit des Algorithmus

Da innere Hindernisse erkannt werden und deren Eingänge geschlossen werden, wird der Roboter immer den inneren Außenrand eines Gebietes finden. Auf diesem inneren Außenrand werde alle Einbahnstraßen gefunden. Nach und nach werden alle diese Einbahnstraßen ausprobiert. Eine dieser wichtigen Einbahnstraßen ist der Ausgang des Labyrinths.
Es kann durchaus vorkommen, dass der Roboter wieder zurück zu einem schon besuchten Gebiet kommen muss, weil z.B. in diesem Gebiet die letzte noch nicht getestete Einbahnstraße liegt. Da der Roboter über eine Liste verfügt, die ihm angibt, welche Einbahnstraßen genommen werden müssen um zu diesem Gebiet zu gelangen, ist dies kein Problem. Hierfür muss der Roboter auch nicht den Pledge-Algorithmus benutzen, da alles diese Einbahnstraßen die von einem Gebiet in das andere führen, allein durch die Wandverfolgung getroffen werden. Welche der noch nicht besuchten Einbahnstraßen als nächstes exploriert wird, kann auf verschiedene Weise bestimmt werden. Hier ist ein Backtrackin Algorithmus möglich, genauso gut kann eine beliebeige andere Graphendurchmusterung für gerichtete Graphen benutzt werden.

3.3.3 Backtracking-Pledge

Der Backtracking-Pledge sucht mit Hilfe eines Backtracking Algorithmus nach der richtigen globalen Verhaltensweise. An den Einbahnstraßen werden Winkelzählerwerte gespeichert und zurückgesetzt um zu garantieren, dass der Roboter bei der Suche immer den gleichen Weg geht. Die Information die durch die Erkennung der Einbahnstraßen gewonnen werden, werden hier genutzt um zu erkennen, wann der Algorithmus in eine Schleife geraten ist und die nächste Möglichkeit im „Suchbaum“ gewählt werden muss.

Beschreibung des Algorithmus

Zur Erinnerung: Eine globale Verhaltensvorschrift ist eine Abbildung, die jeder Einbahnstraße den Zustand *offen* oder *geschlossen* zuordnet. Es gibt also bei einem Labyrinth L_{ein} mit N Einbahnstraßen 2^N verschiedene globale Verhaltensvorschriften. Wenn der Roboter also von seinem Startpunkt aus alle 2^N Möglichkeiten ausprobieren würde, käme er mit Sicherheit bei mindestens einer Verhaltensvorschrift ans Ziel. Dies wäre dann praktikabel, wenn der Roboter eine Möglichkeit hat, seinen Startpunkt wiederzufinden.
Die Anforderung, die an den Einbahnstraßen-Pledge gestellt wird, ist ein ähnlicher Speicherbedarf und eine ähnlich einfache Steuerung wie die des Pledge-Algorithmus. Die Fähigkeit einen Startpunkt wiederzufinden (z.B. über eine Karte) widerspricht aber dieser Anforderung. Also muss die Idee sein, sich an schon vorhandenen, bekannten

Punkten zu orientieren. Hier bieten sich die Einbahnstraßen an.
Der Algorithmus verwendet eine *Verhaltensliste*. Die Verhaltensliste ist eine Liste in der die besuchten Einbahnstraßen und das Verhalten an ihnen abgespeichert wird. Zusätzlich muss zu jeder Einbahnstraße auch der Winkelzählerwert abgespeichert werden, mit dem die Einbahnstraße das erste Mal besucht wurde. Ein Element in der Verhaltensliste ist also folgendes Tripel:

$$\{\textit{Name der Einbahnstraße, Verhalten, Winkelzählerwert}\}$$

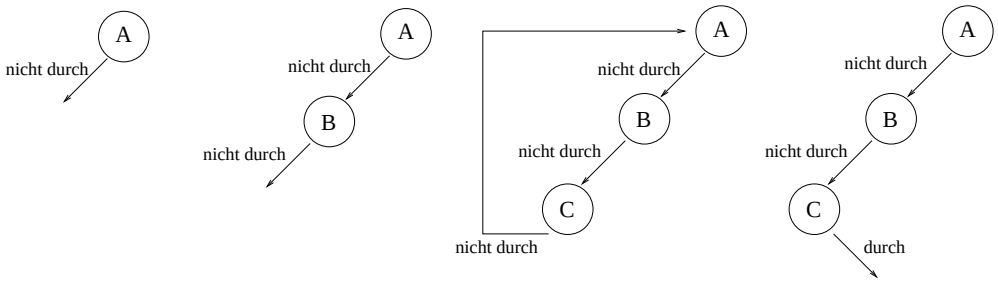

Abbildung 3.14: *Die Entwicklung der Verhaltensliste. Es werden so lange neue Einbahnstraßen eingefügt, bis zu einer bekannten Einbahnstraße zurück gekehrt wurde. Dann wird die Entscheidung auf der untersten Ebene geändert.*

Der Algorithmus startet mit einer leeren Verhaltensliste. Er führt den Pledge-Algorithmus aus, bis er auf eine Einbahnstraße trifft. Da dies die erste Einbahnstraße ist, ist sie garantiert noch nicht in der Verhaltensliste enthalten und wird entsprechend eingefügt. Zusätzlich muss der Algorithmus wissen, wie er sich an dieser Einbahnstraße zu verhalten hat, sprich, ob er durchfahren soll oder nicht. Nach Vereinbarung werden alle neuen Einbahnstraßen als erstes *nicht durchfahren*. Diese Einbahnstraße wird als letzte besuchte Einbahnstraße abgespeichert (im Folgenden *letzte_Ein* genannt).
Wenn der Roboter während seiner Suche nach dem Ausgang an eine Einbahnstraße kommt, gibt es zwei Möglichkeiten:

1. *Die Einbahnstraße ist nicht in der Verhaltensliste.* Dann wird am Ende der Liste ein neuer Eintrag angefügt, letzte_Ein wird aktualisiert und es wird nicht durch diese Einbahnstraße gefahren.

2. *Die Einbahnstraße ist in der Verhaltensliste enthalten.* Jetzt kommt es auf *letzte_Ein* an:

(a) *letzte_Ein ist nicht direkter Vorgänger der aktuellen Einbahnstraße.* Dies bedeutet, der Roboter ist im Kreis gelaufen. letzte_Ein wird aktualisiert. Zusätzlich wird das Verhalten des letzten Eintrags in der Verhaltensliste geändert. Auch hier gibt es wieder zwei Möglichkeiten:

i. *Das Verhalten steht auf „nicht durchgehen“*. Dann wird das Verhalten auf „durchgehen“ gesetzt.

ii. *Das Verhalten steht auf „durchgehen“*. Dann scheinen beide Verhaltensweisen für die Einbahnstraße am Ende der Verhaltensliste in der derzeitigen Situation nicht zum Ziel zu führen. Also wird diese Einbahnstraße aus der Verhaltensliste gelöscht und der Schritt a) erneut durchgeführt, unabhängig davon, ob letzte_Ein nun Vorgänger ist, oder nicht. Und ebenfalls unabhängig davon, ob dadurch die Einbahnstraße gelöscht wird, an der der Roboter gerade steht.

(b) *letzte_Ein ist zu der aktuellen Einbahnstraße direkter Vorgänger in der Verhaltensliste.* Dann verhält sich der Roboter so, wie in der Verhaltensliste für diese Einbahnstraße gespeichert. letzte_Ein wird aktualisiert. Auf diese Weise kommt der Roboter wieder an die Einbahnstraße, an der eine Verhaltensänderung vorgenommen wurde.

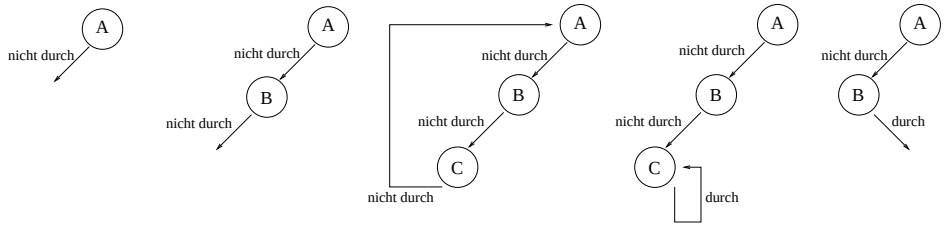

Abbildung 3.15: *Es sind Labyrinthe denkbar, in denen der aktuelle Knoten der Verhaltensliste gelöscht wird. In diesem Beispiel wird C gelöscht, da beide Möglichkeiten (durch und nicht durch) nicht weitergeführt haben. Allerdings befindet sich der Roboter gerade an der Einbahnstraße C, so dass er keine Verhaltensvorgabe für diesen Fall mehr hat.*

Die Entwicklung der Verhaltensliste ist in Abbildung 3.14 zu sehen. Da die Knoten, die sich als nicht ziel führend erwiesen haben, gelöscht werden, ist nur eine einfache Liste zu speichern.
Der Backtracking-Pledge ist ein Backtracking Algorithmus. Im Suchbaum sind die Einbahnstraßen die Knoten. Die Kanten zwischen den Knoten entsprechen den Verhaltensweisen. Von jedem Knoten gehen zwei Kanten ab, diese entsprechen den beiden

Verhaltensweisen *durch* und *nicht durch*.
Da jede Einbahnstraße maximal einmal in der Liste vorkommen kann, enthält die Liste zu jedem Zeitpunkt maximal N Elemente bei N Einbahnstraßen. Für jede Einbahnstraße werden konstant viele Daten gespeichert. So ist der Speicheraufwand für den Backtracking-Pledge in $O(N)$.

Wiederfinden des bekannten Raumes

Wie in Abbildung 3.15 gezeigt, kann die Situation entstehen, dass die aktuelle Einbahnstraße nicht mehr in der Verhaltensliste gespeichert ist. So kann eine Einbahnstraße, z.B. E_i sowohl mit dem Verhalten *nicht durchgehen*, als auch mit dem Verhalten *durchgehen* nicht zum Ziel führen. Das Verhalten *nicht durchgehen* führt zurück in ein Gebiet, aus dem der Ausgang zu finden wäre. Der Algorithmus probiert allerdings nun an der Einbahnstraße E_i das Verhalten *durchgehen* aus. Dieses Verhalten führt aber zurück zu Einbahnstraße E_i. Da beide Verhalten nicht zum Ziel führen, wird der Knoten E_i gelöscht und beim Vorgänger die Verhaltensvorschrift geändert. Allerdings steht der Roboter an der Einbahnstraße E_i und hat in seiner Verhaltensliste kein Verhalten mehr für diese Einbahnstraße.
Der Algorithmus erkennt diese Situation und ruft sich rekursiv auf. Zusätzlich übergibt er der neuen Rekursionsebene einen Zeiger auf die alte Verhaltensliste. Wenn nun die neue Rekursionsebene den Ausgang findet, terminiert der Algorithmus. Der Algorithmus springt eine Ebene zurück, wenn eine erreichte Einbahnstraße in der alten Liste vorhanden ist. So findet der Roboter wieder auf bekannte Einbahnstraßen zurück.
Damit ist sichergestellt, dass die gesamte Anzahl an Einträgen in der Verhaltensliste und den alten Listen gleich der Anzahl an Einbahnstraßen ist. Der Algorithmus hat also weiterhin einen Speicherbedarf in $O(N)$ wobei N die Anzahl der Einbahnstraßen ist.

Dass die Verhaltensliste wirklich eine Liste bleibt und sich nicht zu einem Baum von Entscheidungen entwickelt, wird dadurch garantiert, dass bei gleicher Verhaltensweise an einer Einbahnstraße eine bestimmte nächste Einbahnstraße erreicht wird. Dies wird dadurch ermöglicht, dass der Winkelzählerwert an einer bekannten Einbahnstraße auf den gespeicherten Wert zurückgesetzt wird. Ebenso muss betrachtet werden, an welcher Stelle der Roboter eine Einbahnstraße überquert.

Algorithm 3 Einbahnstraßen Pledge

```
1:  procedure EINBAHNSTRASSEN PLEDGE(alteListe)
2:      Initialisiere Liste mit NULL
3:      Initialisiere LastX mit NULL
4:      mache Pledge bis Einbahnstraße X                ▷ Mit Algorithmus 2
5:      {
6:      if X ∈ alteListe then        ▷ X ist in der vorherigen Liste schon mal besucht
    worden
7:          return X   ▷ Gehe Rekursionsschritt zurück, mit der neuen Einbahnstraße
    X
8:      end if
9:      if X ∉ Liste then
10:         Hänge { X , Winkelzähler , Auswahl: „nicht durch“ } an Liste
11:     end if
12:     if X nicht Nachfolger LastX then
13:         Change (LastX,Liste)
14:     end if
15:     if X ∉ Liste then
16:         X ← EinbahnstraßenPledge(Liste + alteListe)
17:     end if
18:     LastX ← X
19:     hole Element der Liste mit X
20:     setze Roboter Winkelzähler auf Winkelzähler
21:     gehe Auswahl                ▷ Anweisung ob Pledge durchgehen soll oder nicht
22:     }
23: end procedure
24:
25:
26: procedure CHANGE(Einbahnstraße,Liste)
27:     if Auswahl von Einbahnstraße = „nicht durch“ then
28:         Auswahl von Einbahnstraße ← „durch“
29:     else
30:         if X nicht erstes Element then
31:             Change( Vorgänger X)
32:         end if
33:         lösche X
34:         return
35:     end if
36: end procedure
```

Korrektheit des Algorithmus

Nach Lemma 3.5 existiert ein Weg zum Ausgang, so dass der Roboter nur maximal einmal durch jede Einbahnstraße gehen muss. Allerdings kann es sein, dass der Roboter durch eine Einbahnstraße mehrere Male *nicht durchgehen* muss. So z.B. wenn der Roboter innerhalb einer Spirale startet und der Eingang mit Einbahnstraßen versehen ist (vgl. Abb. 3.16). So kann es also sein, dass der Backtracking Pledge ein Kreis gefunden hat und die Verhaltensweise ändert, obwohl die richtige globale Verhaltensweise bereits gefunden wurde. Ein solches „mehrfach nicht durchfahren“ geschieht immer dann, wenn der Roboter an einem inneren Hindernis entlangfährt und sein Winkelzählerwert echt kleiner -2π ist. In diesem Abschnitt wird gezeigt, dass der Roboter herausfindet, obwohl der Algorithmus möglicherweise nicht beim ersten Mal durch die richtige globale Verhaltensweise herausfindet.

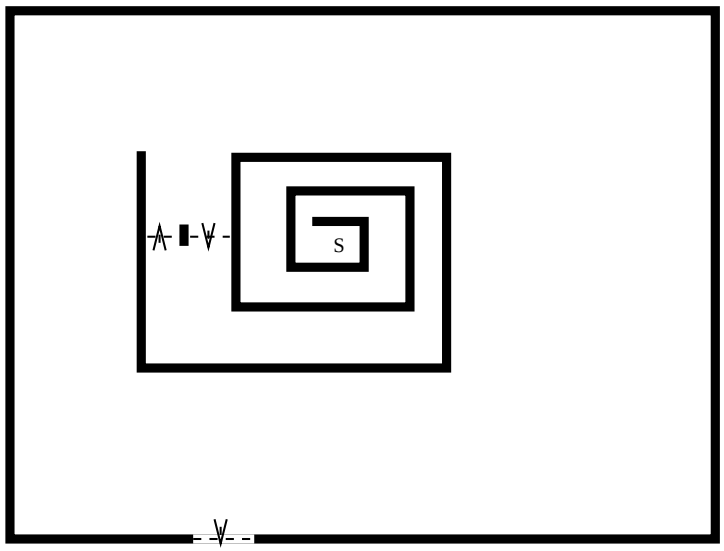

Abbildung 3.16: *Wenn der Roboter im Inneren der Spirale startet, wird der Winkelzähler klein genug, dass der Roboter zweimal an der Einbahnstraße, die in die Spirale hineinführt, vorbeikommt. Damit der Roboter nicht mehrfach durch die aus der Spirale führenden Einbahnstraße fahren muss, ist es sinnvoll, dass der Roboter zweimal nicht durch die Einbahnstraße fährt.*

Theorem 3.29 *: Der Backtracking-Pledge findet immer einen Weg aus einem fairen Labyrinth mit Einbahnstraßen.*

Um dieses Theorem zu beweisen, wird ersteinmal angenommne, dass der Roboter alle wichtigen Ausgänge findet. In den Labyrinthen kann es vorkommen, dass der Algorithmus eine richtige globale Verhaltensweise nicht sofort erkennt. Dies liegt wie oben gesagt daran, dass der Winkelzähler des Roboters zu hoch ist. Wie man jedoch sieht, wird der Algorithmus, nachdem er jede Verhaltensweise ausprobiert und verworfen hat sobald er einen Kreis entdeckt, neu gestartet. Jedoch hat sich der Winkelzähler im Gegensatz zum letzten Start vergrößert. So bringt der Algorithmus den Roboter wenn es notwendig ist, nach und nach in eine Position, wo nicht nur jede Einbahnstraße nur einmal durchlaufen werden muss, sonder auch jede Einbahnstraße nur einmal besucht werden muss um den Ausgang zu finden.

Lemma 3.30*: Der Backtracking-Pledge entfernt sich beliebig weit von einer Struktur S aus Gebieten, wenn er alle wichtigen Einbahnstraßen findet und S mindestens einen Ausgang hat.*

Beweis. Angenommen, der Roboter würde sich nicht von der Struktur S entfernen.
Das der Roboter auch außerhalb von S fährt, ist dadurch gegeben, dass der Roboter die Ausgänge aus S findet und mit Hilfe des Backtracking auch durchfährt. Damit der Roboter sich nicht von der Struktur S entfernt, muss er außerhalb von S einen Winkelzähler haben, der immer kleiner Null ist. Sollte der Roboter ausserhalb von S den Winkelzählerwert von Null erreichen, so würde er sich vom Hinderniss lösen. Dies bedeutet, dass keine der Einbahnstraßen die aus S führt, mit einem Winkelzähler größer oder gleich -2π initialisiert werden darf.
Da der Roboter immer wieder durch die Einbahnstraßen fährt existiert ein Zyklus von Verhaltensweise, der sich immer wiederholt. Sei E die Einbahnstraße, mit der dieser Zyklus anfängt. Von ihr aus werden alle möglichen Kombination der Einbahnstraßen aus offen und geschlossen ausprobiert. So lange, bis E selbst wieder gelöscht wird. Dann beginnt es von vorne und E wird neu initialisiert.
Nun wird die Entwicklung des Winkelzählers bei dieser Initialisierung betrachtet. Bevor E gelöscht wird, wird durch den Algorithmus als letzte Möglichkeit ausprobiert, dass durch alle erreichbaren Einbahnstraßen durchgegangen wird. Da der Roboter in einer Struktur ist Hindernis ist und durch alle Einbahnstraßen durchgeht, vollführt er eine geschlossene Kurve von E wieder zu E gegen den Uhrzeigersinn. Eine geschlossene Kurve im Uhrzeigersinn ist nicht möglich, da der Roboter sich dann in einem Innehof befinden würde und das Labyrinth somit nicht lösbar ist. So wird der Winkelzähler zur erneuten Initialisierung von E um 2π größer. Dies bedeutet, dass wenn dieser Zyklus erneut durchgeführt wird, der Winkelzähler sich immer weiter erhöht. Hieraus folgt, dass E kein Ausgang aus S sein darf, da sich sonst der Roboter von S entfernt. Es wird also nach einiger Zeit E nicht mehr getroffen. Es kann keine Einbahn-

straße getroffen werden, die vor E im Suchbaum liegt, da angenommen wurde das mit E der Zyklus beginnt. Wird jedoch eine Einbahnstraße getroffen, die im Suchbaum nach E liegt, so beginnt dort ein neuer Zyklus. Da es nur endlich viele Einbahnstraßen im Labyrinth geben kann, wird irgendwann ein Ausgang aus S zum Beginn des Zyklus und damit irgendwann mit einem Winkelzähler größer als -2π initialisiert.
$\longrightarrow$ Widerspruch

■

Aus dem Lemma 3.30 ergibt sich folgendes Lemma:

Lemma 3.31*Der Backtracking-Pledge entfernt sich beliebig weit von einem Labyrinth L aus Einbahnstraßen, welches nur ein Gebiet hat, wenn er alle Einbahnstraßen findet.*

Beweis. Wie vorher gezeigt, entfernt sich der Roboter beliebig weit von einem Labyrinth, dass aus mehreren Gebieten besteht. Wird num um das L eine Bounding Box aus Einbahnstraßen gelegt, die alle von L wegführen, so entfernt sich der Roboter vonn diesem Gebielde beliebeig weit, also auch von L. Da man diese Bounding Box beliebig weit von L weglegen kann, entfernt sich der Roboter auch ohne diese Boundig Box beliebig weit von L. ■

Lemma 3.32*: An welchem Punkt der Roboter die Einbahnstraße überfährt, ändert nichts an der Lösbarkeit des Labyrinths.*

Beweis. Es wird gezeigt, dass es kein Labyrinth gibt, bei dem es wichtig ist, an welchem Punkt eine Einbahnstraße überfahren wird.
Gegeben ist das Labyrinth L_{ein}.
Es gibt nur zwei Möglichkeiten für den Roboter an eine Einbahnstraße zu gelangen:

1. Durch eine freie Bewegung im Raum, d.h. der Winkelzähler ist Null und der Roboter folgt keiner Wand.

2. Der Winkelzähler ist nicht Null und der Roboter befindet sich im „Wall-Following Modus“. Dies bedeutet der Roboter trifft auf ein Ende der Einbahnstraße.

Im ersten Fall ist es möglich, dass der Roboter jeden Punkt der Einbahnstraße durchlaufen könnte. Allerdings ist der Winkelzähler Null. Dies bedeutet, man kann den Durchfahrtspunkt als neuen Startpunkt des Algorithmus betrachten. Also kann eine Verschiebung des Roboter auf der Einbahnstraße keinen Einfluss auf die Lösbarkeit haben, da es sonst einen Startpunkt gäbe, von dem der Roboter nicht aus dem Labyrinth finden würde. Wenn nun, wie später gezeigt wird, der Backtracking-Pledge aus dem Labyrinth findet, ist es egal, an welcher Stelle der Roboter bei einer freien Bewegung die Einbahnstraße durchfährt.

Im zweiten Fall kann der Roboter eine Einbahnstraße nur an einem der Enden der Einbahnstraße treffen. Der Roboter trifft bedingt dadurch, dass er die Wand immer auf der linken Seite hat, immer auf das linke Ende der Einbahnstraße.

■

Es muss betrachtet werden, aus welchen Richtungen der Roboter auf eine Einbahnstraße trifft, da für den Algorithmus angenommen wird, dass bei gleicher Verhaltenswahl der gleiche Weg zurückgelegt wird. So ist es durchaus wichtig, wie der Roboter auf eine Einbahnstraße treffen kann.
Wie in Lemma 3.32 gezeigt, kann der Roboter nur bei einer freien Bewegung auf einen beliebigen Punkt der Einbahnstraße treffen. Da es allerdings für freie Bewegungen keinen Unterschied macht, wo der Roboter durch die Einbahnstraße fährt (vgl. Lemma 3.32), kann für den Algorithmus angenommen werden, dass der Roboter so gesteuert wird, dass er immer durch das linke Ende der Einbahnstraße fährt. So ist garantiert, dass, bei gleicher Verhaltenswahl, der gleiche Weg zurückgelegt wird.

Lemma 3.33*: Wenn der Roboter den Außenrand eines Gebietes trifft, werden alle Einbahnstraßen die auf dem Rand dieses Gebietes liegen, gefunden.*

Beweis. Angenommen, der Roboter trifft nach einer freien Bewegung auf den inneren Rand. Dies bedeutet, dass der Winkelzähler zum Zeitpunkt des Auftreffens gleich Null ist. Um dem inneren Rand zu folgen muss der Roboter sich im Uhrzeigersinn drehen, dadurch wird der Winkelzähler kleiner Null. Da, nach Vereinbarung des Algorithmus, jede Einbahnstraße erst *nicht durchfahren* werden soll, kann der innere Rand des Gebietes als eine einfache, geschlossene polygonale Kette angesehen werden. Dies bedeutet, dass der Roboter dieser Kette im Uhrzeigersinn folgt. Der Winkelzähler kann hierdurch nicht wieder gleich Null werden. Hieraus folgt, dass der Roboter mindestens einmal das gesamte Gebiet innen umkreist und somit alle Einbahnstraßen findet. ■

Um zu zeigen, dass der Algorithmus jedes Labyrinth löst, muss gezeigt werden, dass alle Einbahnstraßen gefunden werden, welche für den Weg zum Ausgang notwendig sind.

Lemma 3.34*: Alle wichtigen Ausgänge werden gefunden.*

Beweis. Angenommen es gibt wichtige Ausgänge, die nicht gefunden werden. Dann existiert vom Standpunkt des Roboters aus, ein erster wichtiger Ausgang. Sei dieser erste Ausgang die Einbahnstraße E. E ist ein wichtiger Ausgang aus dem Gebiet G.
Es existieren nun zwei Möglichkeiten, wo sich der Roboter befinden kann.

- Der Roboter befindet sich außerhalb von G. Dann findet der Roboter allerdings einen Eingang nach G, da E ja der erste nicht gefunden Ausgang ist. Durch das

Backtracking Verfahren ist sichergestellt, dass der Roboter nach G einfährt. Also kann angenommen werden, dass der Roboter sich in G befindet.

- Der Roboter befindet sich in G. Wie oben gezeigt, ist dies der einzige interessante Fall.

Damit E nicht gefunden wird, darf der Roboter nicht an den Rand von G kommen (vgl. Lemma 3.33). Die kann nur geschehen, wenn der Roboter im Inneren von G an einem inneren Hindernis S bleibt. Dieses innere Hindernis muss aus mindestens einem Gebiet bestehen (vgl. Lemma 3.31). Jedoch löst sich der Roboter von diesem inneren Hindernis (vgl. Lemma 3.30). Der Roboter muss also nach dem Lösen von S wieder auf ein innere Hindernis stoßen. Auch von diesem Hindernis löst sich der Roboter. Dieses Lösen geschieht immer in die gleiche Richtung (die freie Bewegung geschieht durch die Pledge-Steuerung immer in eine bestimmte Richtung). Es müssen also unendliche viele innere Hindernisse in G existieren, damit der Roboter nicht an den Rand von G kommt. Die ist aber ein Widerspruch gegen die Definition eines Labyrinthes.
der Roboter findet den Rand von G und E.
$\longrightarrow$ Widerspruch

■

Nun sind alle Voraussetzungen gegeben, um Theorem 3.29 zu beweisen.

Beweis. Dies zeigt, dass es keine wichtige Einbahnstraße gibt, die nicht gefunden wird. Also werden alle Einbahnstraßen, die der Ausgang sein könnten auch in die Verhaltensliste aufgenommen. Da bei einem fairen Labyrinth bei alle Einbahnstraßen jede Möglichkeit ausprobiert wird (zumindest so weit, bis der Ausgang gefunden wurde), findet der Backtracking-Pledge immer den Ausgang aus einem unbekannten Labyrinth mit Einbahnstraßen.

■

Beispiele für Fahrten des Bachtracking-Pledge

Schon beim Beweis der Korrektheit des Backtracking-Pledge (siehe Theorem 3.29) zeigt sich, dass es zwei Arten von Labyrinthen gibt:

1. Labyrinthe ohne ineinander verschachtelte Gebiete
2. Labyrinthe mit ineinander verschachtelten Gebieten

Abbildung 3.17 ist ein Beispiel für die 1. Labyrinth-Art. Hier sind verschiedene Gebiete so aneinander gelegt, dass kein Gebiet vollständig innerhalb eines anderen Gebietes liegt. Ebenso gibt es innerhalb eines Gebietes keine Hindernisse. In Abbildung 3.18 bis 3.25 wird die Abfolge der Fahrt des Einbahnstraßen-Pledge in diesem Beispiellabyrinth gezeigt.

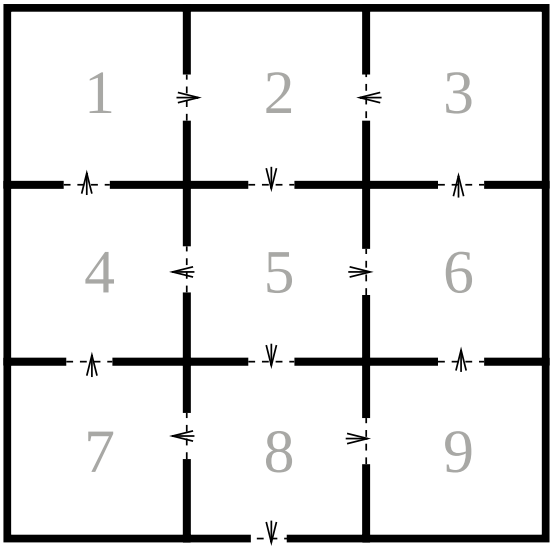

Abbildung 3.17: *In diesem Labyrinth existiert kein Gebiet, welches von einem anderen eingeschlossen ist.*

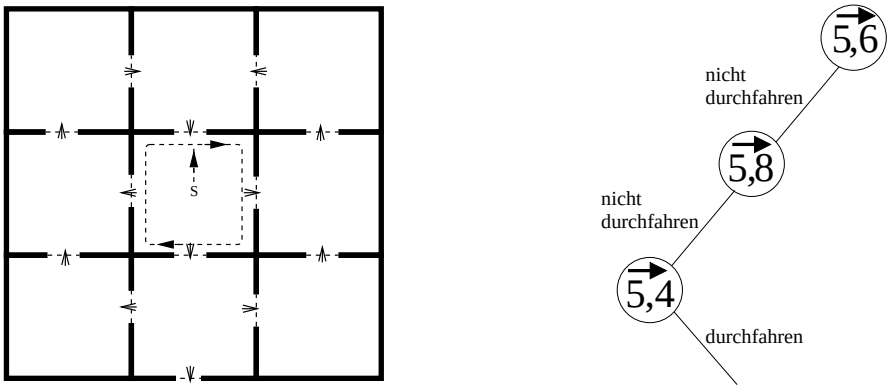

Abbildung 3.18: *Der Roboter startet in der Mitte des Labyrinths am Startpunkt S. Er fährt so lange nach vorne, bis er auf ein Hindernis trifft. Er folgt der Umrandung des Gebietes 5 (vergl. Gebietsnummer mit Abbildung 3.17). Hierbei werden alle Einbahnstraßen, die aus diesem Gebiet herausfahren in die Verhaltensliste eingefügt. Die Einbahnstraße $\overrightarrow{5,6}$ (von Gebiet 5 nach Gebiet 6 führend) wird als Erste eingefügt, danach die Einbahnstraße $\overrightarrow{5,8}$ und zuletzt $\overrightarrow{5,4}$. In der zweiten Runde im Gebiet 5 wird diese zuletzt eingefügte Einbahnstraße $\overrightarrow{5,4}$ geöffnet.*

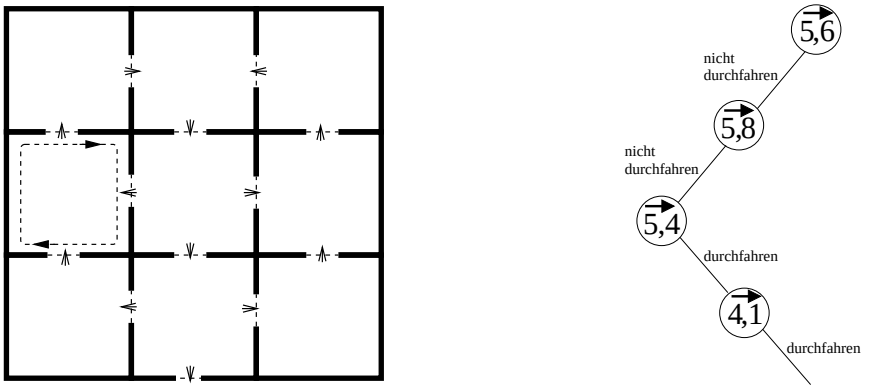

Abbildung 3.19: *Der Roboter ist in das Gebiet 4 eingefahren. Auch hier wird wieder die gesamte Umrandung abgefahren. Man sieht schon hier, dass der Roboter innerhalb des Labyrinthes nie mehr den Winkelzähler Null erreicht. Der Roboter wird immer weiter der Wand folgen. Wenn der Roboter das zweite Mal an der Einbahnstraße $\overrightarrow{4,1}$ vorbei fährt, ist diese offen und er fährt durch.*

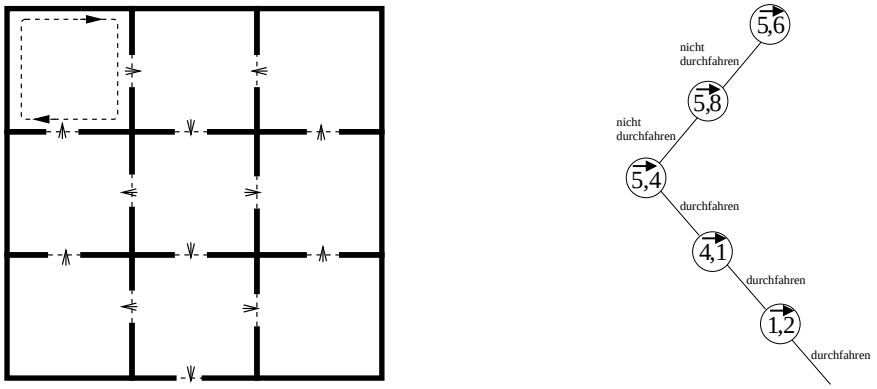

Abbildung 3.20: *Wie im vorherigen Gebiet, wird auch hier erst die neue Einbahnstraße in die Verhaltensliste aufgenommen. Beim zweiten Umfahren steht diese Einbahnstraße* $\overrightarrow{1,2}$ *auf* durchgehen.

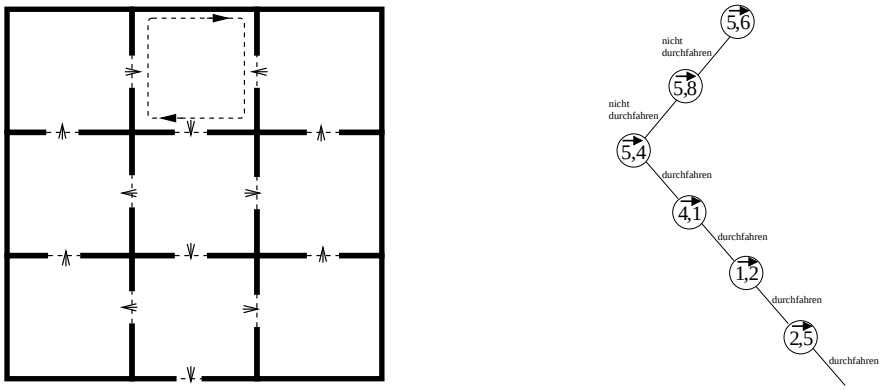

Abbildung 3.21: *Auch hier kreist der Roboter einmal im Gebiet, findet die Einbahnstraße* $\overrightarrow{2,5}$ *und durchfährt diese beim zweiten Auftreffen.*

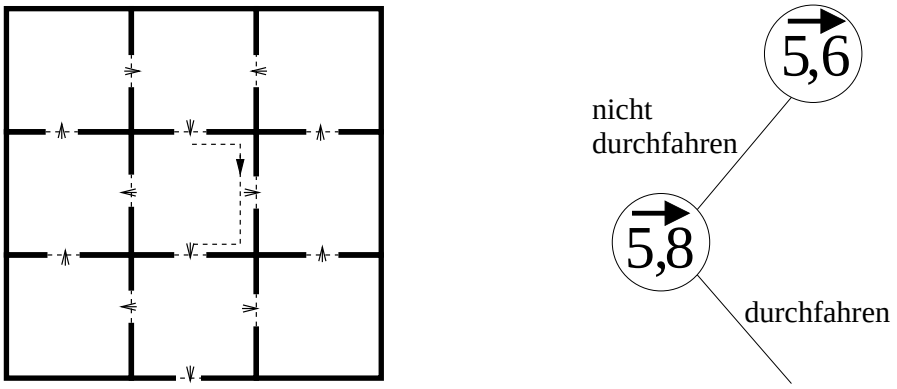

Abbildung 3.22: *Der Roboter ist wieder zurück im Gebiet 5. Hier trifft er wieder auf die Einbahnstraße* $\overrightarrow{5,6}$*. Diese Einbahnstraße ist in der Verhaltensliste vorhanden. Also wird das Verhalten an der letzten gespeicherten Einbahnstraße geändert. Dies ist die Einbahnstraße* $\overrightarrow{2,5}$*. Diese wird gelöscht, da sie schon auf* durchgehen *steht. Zusätzlich wird das Verhalten der jetzt letzten gespeicherten Einbahnstraße geändert. Auch diese wird gelöscht. Dies geht so weit, bis die Einbahnstraße* $\overrightarrow{5,8}$ *geöffnet wird. Der Roboter muss also dieses Mal nicht komplett das Gebiet 5 umrunden.*

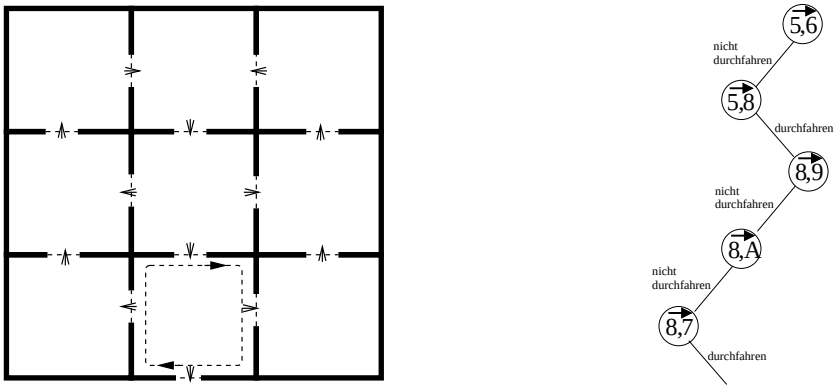

Abbildung 3.23: *Der Roboter fährt von Innen die Außenwand des Gebietes 8 ab. Hierbei wird der Ausgang (die Einbahnstraße* $\overrightarrow{8,A}$*) in die Verhaltensliste eingefügt, allerdings noch nicht durchlaufen.*

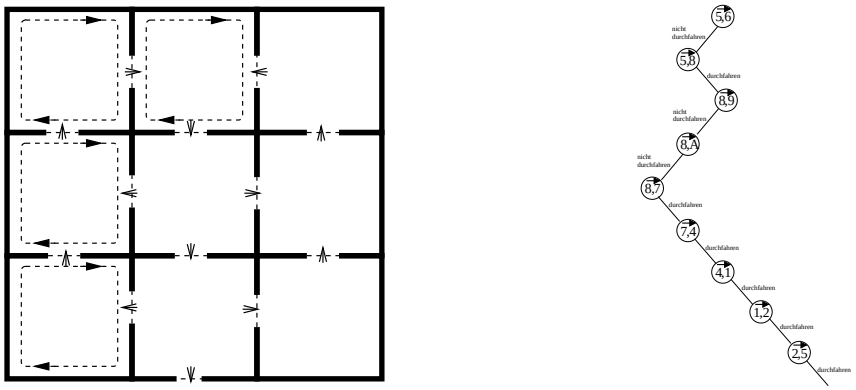

Abbildung 3.24: *Der Roboter fährt in das Gebiet 7 ein und nimmt dann erneut den Weg über 4, 1 und 2.*

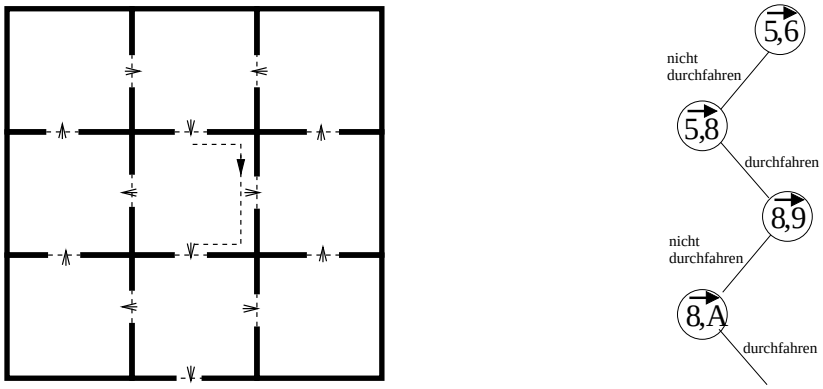

Abbildung 3.25: *Erneut erreicht der Roboter das Gebiet 5. Dieses Mal wird bei der Aktualisierung die Einbahnstraße des Ausgangs auf* durchgehen *gesetzt.*

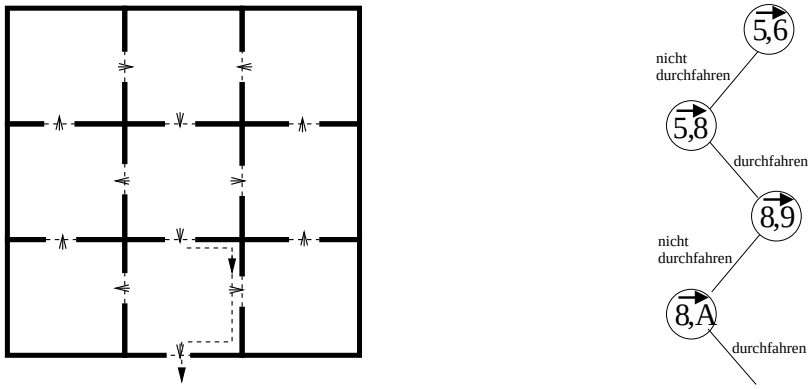

Abbildung 3.26: *Der Roboter ist wieder in das Gebiet 8 eingefahren. Da der Ausgang nun auf* durchgehen *steht, verlässt der Roboter das Labyrinth.*

Abbildung 3.27 zeigt ein Beispiel für Labyrinthe der 2. Art. Hier ist das Innere der Spirale ein Gebiet, welches vollständig im äußeren Gebiet liegt. Dieses äußere Gebiet enthält den Ausgang. Die Abbildungen 3.28 bis 3.34 zeigen die Fahrt des Roboters durch dieses Labyrinth. Wie oft der Roboter in die Spirale wieder einfährt, hängt davon ab, wie viele Windungen diese besitzt.

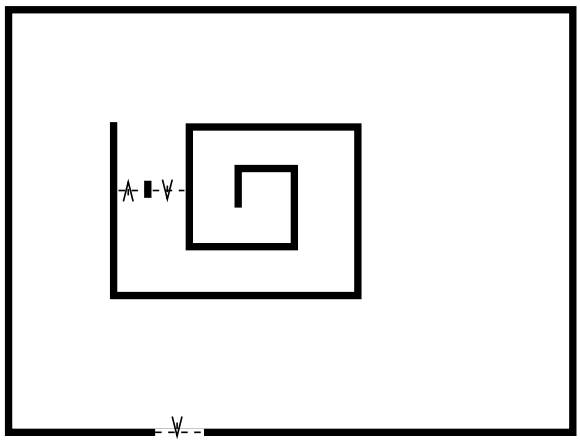

Abbildung 3.27: *In diesem Labyrinth existiert ein Gebiet, welches von einem anderen Gebiet umschlossen ist.*

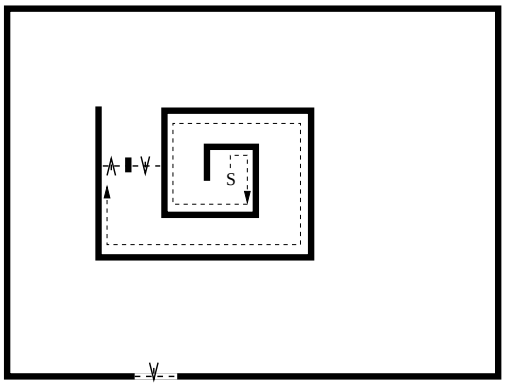

Abbildung 3.28: *Der Roboter startet in der Mitte der Spirale mit dem Winkelzähler 0 und einer leeren Verhaltensliste. Bis zur Einbahnstraße ist der Weg eine einfache Pledge-Steuerung. Der Roboter trifft an die Einbahnstraße mit einem Winkelzählerwert von* $2 \cdot (-2\pi)$*. Für diese Einbahnstraße wird dieser Wert gespeichert und die Einbahnstraße wird auf* geschlossen *gesetzt.*

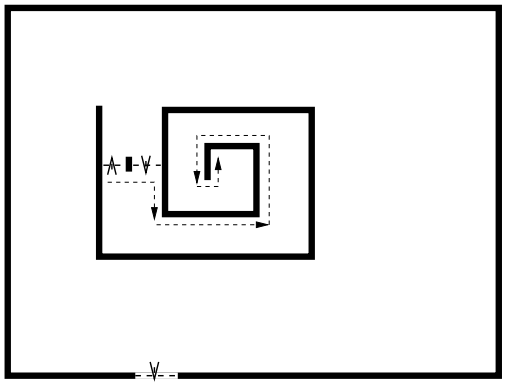

Abbildung 3.29: *Der Roboter fährt wieder bis zur Mitte. Hier hat der Winkelzähler noch einen Wert von* (-2π)*. Die zweite Einbahnstraße an der der Roboter vorbeigekommen ist, wird von ihm nicht wahrgenommen, da er von der* verbotenen Seite *herangefahren ist.*

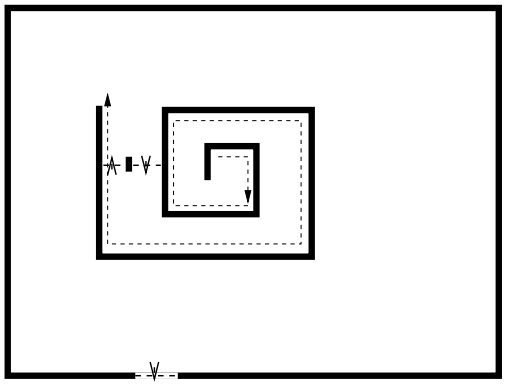

Abbildung 3.30: *Der Roboter fährt wieder aus der Spirale heraus. Wenn er nun an die Einbahnstraße kommt, so ist diese schon in der Verhaltensliste. Da es das letzte (einzige) Element dieser Verhaltensliste ist, wird das Verhalten des Roboters für diese Einbahnstraße auf* durchgehen *gesetzt. Ebenso wird der Winkelzähler auf den gespeicherten Wert zurückgesetzt. Der Winkelzählerwert beträgt nun also wieder* $2 \cdot (-2\pi)$.

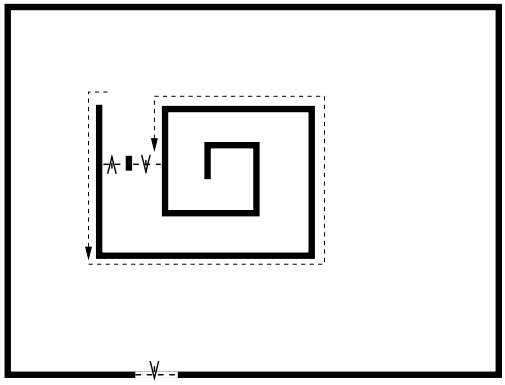

Abbildung 3.31: *Der Roboter fährt einmal um die gesamte Spirale herum. Wenn er auf die zweite Einbahnstraße trifft, hat er noch einen Winkelzählerwert von* π*. Die Einbahnstraße war bis jetzt unbekannt, also wird sie, mit dem Winkelzählerwert und der Verhaltensweise* nicht durchgehen *in der Verhaltensliste gespeichert.*

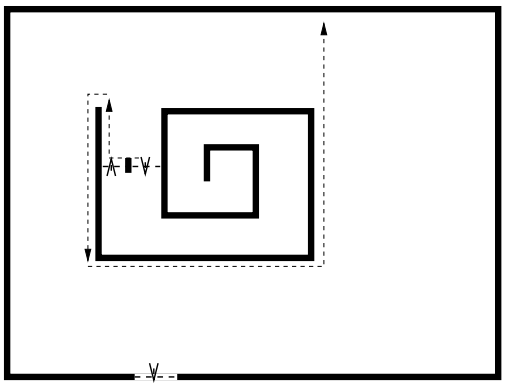

Abbildung 3.32: *Der Roboter ist nicht durch die Einbahnstraße hinein gegangen, und umfährt nochmal die Spirale. Hierbei wird der Winkelzähler allerdings gleich Null und der Roboter löst sich von der Spirale.*

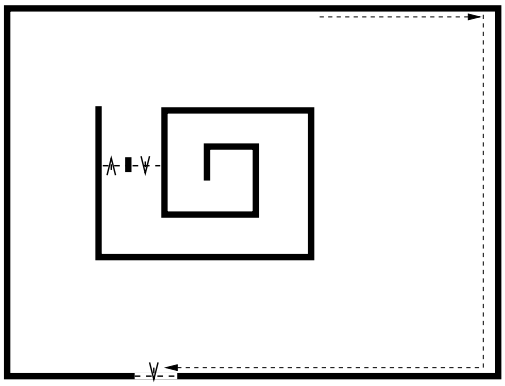

Abbildung 3.33: *Der Roboter folgt der Außenwand. Dabei wird der Winkelzählerwert kleiner. Die Einbahnstraße, die den Ausgang markiert, wird in die Verhaltensliste aufgenommen und mit* nicht durchfahren *markiert.*

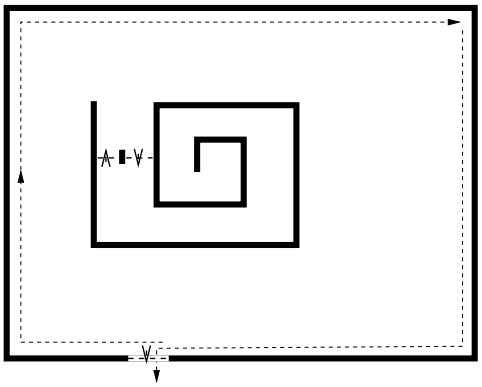

Abbildung 3.34: *Beim ersten Vorbeifahren wird der Ausgang noch nicht gefunden. Der Roboter fährt weiter an der Außenwand entlang. Die Einbahnstraße am Ausgang wird das zweite Mal erreicht. Da sie die Einbahnstraße ist, die als Letzte in die Verhaltensliste eingefügt wurde, wird bei ihr die Verhaltensweise umgestellt. Also fährt der Roboter jetzt durch die Einbahnstraße und hat das Labyrinth verlassen.*

Kapitel 4

Implementierung

In diesem Kapitel wird die praktische Umsetzung des Backtracking-Pledge auf dem Khepera II beschrieben. Hierfür müssen einige Vorarbeiten geleistet werden. So muß eine sinnvolle Testwelt gebaut werden. Ebenso muß der Pledge-Algorithmus als Grundlage auf dem Khepera II implementiert werden. Letztendlich wird die Erweiterung des Roboter mit einer Kamera beschrieben und die Implementierung des Backtracking-Pledge beschrieben.

4.1 Die Testwelt

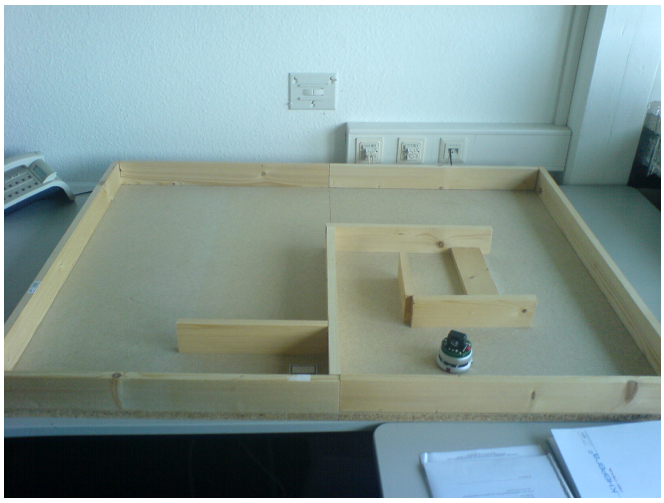

Abbildung 4.1: *Die Testwelt im Überblick.*

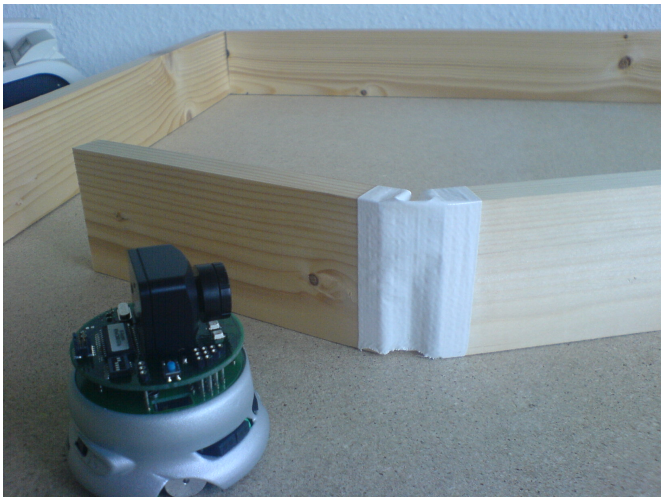

Abbildung 4.2: *Die Realisierung von nicht-rechtwinkligen Hindernissen. Der Spalt zwischen den beiden Holzstücken wird mit Textilband verklebt, so dass sich für die Distanzsensoren des Khepera keine Spalten zeigen. Der Khepera hier ist mit der Kamera ausgestattet. Sie wird in Kapitel 4.3.1 genauer beschrieben.*

Als Spielwelt (aus dem englischen „Toy World“) wird eine konstruierte Umgebung für einen Roboter bezeichnet. In Gegensatz zu realen Umgebungen sind die einzelnen Parameter der Spielwelt kontrolliert und festgelegt. So ist z.B. der Untergrund bekannt und die Steuerung des Roboters kann darauf hin angepasst werden. Andere kontrollierte Parameter können z.B. sein: Höhe von Hindernissen, Gestalt von Hindernissen, Lichtverhältnisse u. s. w.
An die Spielwelt, die für diese Diplomarbeit angefertigt wurde, wurden folgende Bedingungen gestellt:

- Modular: Es müssen verschiedene Labyrinthe aufgebaut werden können. Insbesondere soll sich der Aufbau nicht auf rechtwinklige Labyrinthe beschränken.
- Für den Roboter geeignet: Die Hindernisse müssen eine ausreichende Höhe haben um von den Sensoren des Roboters sehr gut erkannt zu werden.
- Griffige Oberfläche: Da der Pledge-Algorithmus auf Odometrie basiert, muss die Oberfläche so beschaffen sein, dass die Räder des Roboters immer guten Halt haben und Schlupf möglichst vermieden wird.

Als Material wurde Holz verwendet. Die Spielwelt wird auf einer 1245 mm mal 950

mm großen Holzplatte aufgebaut (siehe Abb 4.1). Die Hindernisse bestehen aus Holzlatten mit einer Höhe von 70 mm. Mit diesen Holzlatten lässt sich auch die Umrandung bauen. Diese Holzlatten werden mit Hilfe von Holzdübeln befestigt, so dass der Roboter auch bei einer Fehlsteuerung die Spielwelt nicht verlassen kann.
Hindernisse innerhalb der Spielwelt werden lose aufgestellt. Um Winkel ungleich 90 Grad zu erhalten werden zwei Holzstücke mit Textilband verbunden (ein Beispiel ist in Abb. 4.2 zu sehen). So entstehen keine Lücken an den Ansatzstücken, die die Sensoren des Roboters fehl leiten könnten.

4.2 Khepera II - Der verwendete Roboter

In Kapitel 2 wurde der Khepera II vorgestellt, wie er in der Literatur und in den Handbüchern beschrieben wird. Da es ein reales System ist, welches herstellungsbedingte Schwankungen unterliegt, müssen die angegebenen Werte aus den Handbüchern überprüft und evtl. angepasst werden.

4.2.1 Die Sensorik

Die wohl auffälligste Abweichung des realen Roboters gegenüber der beschriebenen „Idealversion“ ist die Anordnung der Sensoren. In Abbildung 4.3 ist zu sehen, dass die Sensoren an den Seiten nicht im 90° Winkel zur Fahrtrichtung abstrahlen.

Infrarot-Sensoren haben bauartbedingt ein gewisses Rauschen auf den Werten. In Abbildung 4.4 kann man dieses Rauschen erkennen. Die x-Achse zeigt die Entwicklung der Messwerte in der Zeit. Die einzelnen Messwerte sind durchnummeriert. 40 Werte entsprechen einer Sekunde. Die Messreihe ist somit 25 Sekunden lang. Die y-Achse zeigt die Werte, die vom Roboter geliefert werden. Dies sind Werte, die von einem A/D-Wandler erzeugt wurden. Die Eingabe ist der Widerstand über dem IR-Sensor. Die Ausgabe ist eine 10-Bit Integer.
Bei der Aufnahme dieser Zeitreihe wurde der Roboter nicht bewegt. Es befindet sich ein Hindernis auf der linken Seite. Dies sieht man an dem höheren Wert des linken Sensors (rot). Sowohl der Sensor vorne (grün) wie auch der Sensor hinten (blau) registrieren nichts. Hier kann man schon unterschiedliche Ruhewerte für verschiedene Sensoren vermuten.
Anhand dieser Zeitreihen ist das Rauschen zu erkennen. Dies ist zwar ständig vorhanden, hat aber nur eine sehr geringe Varianz.

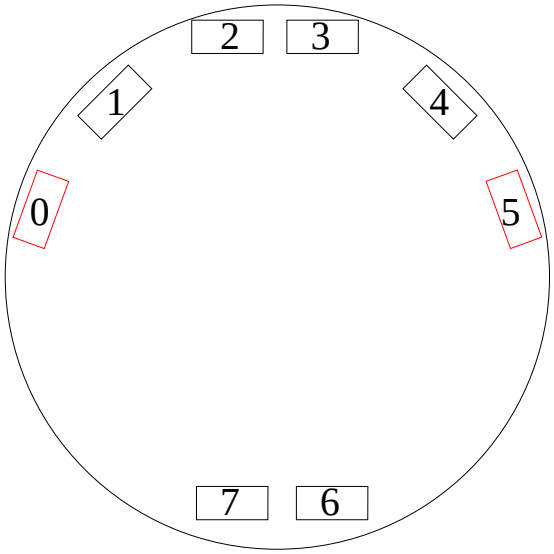

Abbildung 4.3: *Die reale Anordnung der Sensoren des Khepera II Roboters. Die entscheidenden Unterschiede zu der angegeben Konfiguration sind die Sensoren* 0 *und* 5 *(rot markiert).*

Wenn auch dieses Rauschen ungewollt ist, können die Sensoren gefiltert werden. Da das Rauschen eine relative hohe Frequenz hat, und die Änderung der Sensorwerte durch Bewegungen relativ zum Hindernis eher langsam ist, bietet sich hier ein Tiefpass-Filter an. In Abbildung 4.5 sind solche gefilterten Werte dargestellt. Zu Beachten ist, dass durch die Filterung eine Änderung der Werte durch Bewegung an Hindernissen nicht mehr ganz so schnell wahrgenommen werden kann. In Abbildung 4.5 ist dies es zu sehen. Etwa zu dem Zeitpunkt $220/40$ s wurde das Hindernis vor dem Roboter ohne Zeitverzögerung entfernt[1]. Es ist deutlich zu sehen, dass die Sensorwerte eine gewisse Zeit benötigen, um den wirklichen Sensorwert zu erreichen. Es muss also für jede Aufgabe des Roboters ein Kompromiss zwischen Reaktionszeiten und „glatten“ Sensorwerten gefunden werden.

Die Infrarot-Sensoren bilden einen Teil der Umwelt für den Roboter ab. Die Werte, die die Sensoren liefern, entsprechen also Entfernungen zu Hindernissen. Um mit diesen Werten arbeiten zu können, muss klar sein, welcher Sensorwert welcher Entfernung entspricht. Eine Messung hierfür ist in Abbildung 4.6 zu sehen. Auf der x-Achse ist die Entfernung des gemessenen Hindernisses zu sehen. Für diese Messung ist das Hinder-

[1] Es wurde nach oben weggezogen, so, dass die Sensoren schlagartig kein Hindernis mehr sahen

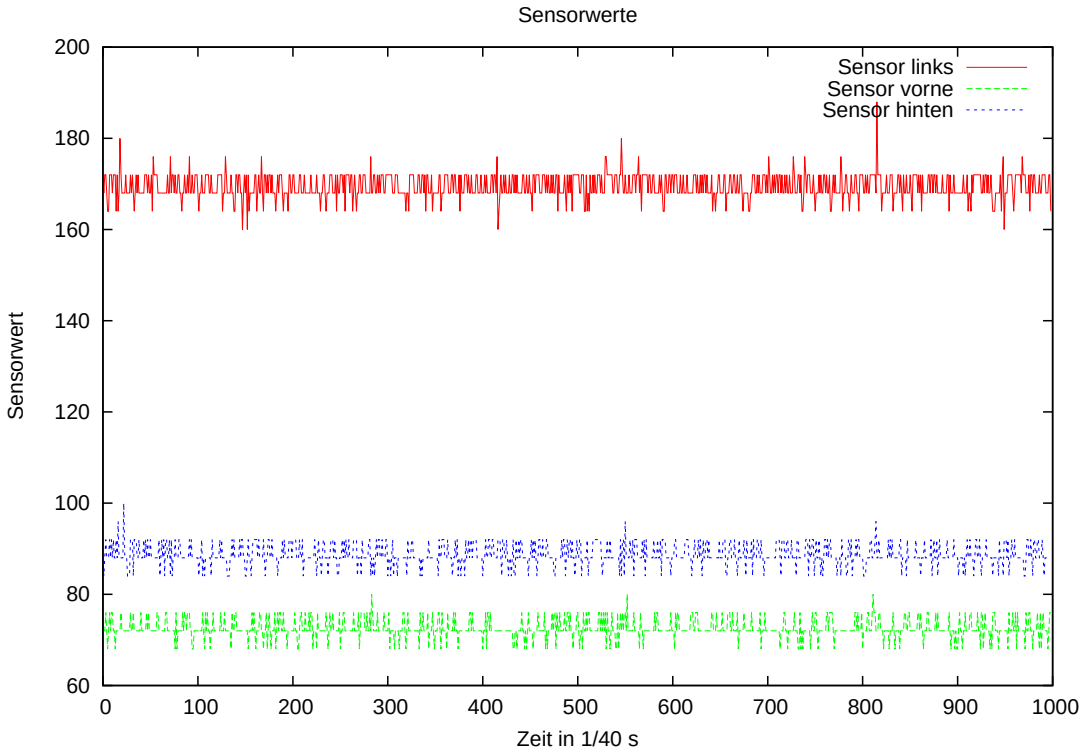

Abbildung 4.4: *Eine Zeitreihe von ungefilterten Werten der Sensoren Nummer 0 (links), Nummer 3 (vorne) und Nummer 7 (hinten). Man sieht ein deutliches, wenn auch geringes, Rauschen auf den Werten.*

nis eine glatte weiße Wand, die senkrecht zu Ausbreitungsrichtung des Infrarotstrahls steht.

Auf der y-Achse ist der Sensorwert aufgetragen. Der maximale Wert, den ein Sensor liefern kann, liegt bei 1024, da im Roboter ein A/D-Wandler mit 10-Bit unsigned den Innenwiderstand des Sensors in digitale Sensorwerte übersetzt. Jeder gemessene Wert der Abbildung, wurde als Mittelwert aus 50 Messungen an diesem Sensor bei gleichbleibender Umwelt eingetragen. So wurde der Einfluss des Rauschens minimiert.
Die wohl wichtigste Erkenntnis der Abbildung ist die Sichtweite der einzelnen Sensoren. Entgegen der im Handbuch angegebenen 10 Zentimeter Sichtweite, ist hier nur eine verlässliche Sichtweite von 8 cm angemessen. Die Unterschiede zwischen der Messung von 8 und 10 cm können leicht im Rauschen untergehen.
Zusätzlich ist zu erkennen, dass ein bestimmter Sensorwert je nach Sensor eine unter-

schiedliche Distanz bedeuten kann. Dies muss bei der Arbeit direkt auf den Sensorwerten bedacht werden. Eine mögliche Umgehung dieses Problems ist die Benutzung einer Look-Up Table mit deren Hilfe eine Sensorwert für einen bestimmten Sensor in eine Entfernung umgerechnet werden kann.

4.2.2 Die Motorik

Um die Motoren anzusteuern wurde die direkte Ansteuerung gewählt. Jedem Motor wird ein Motorwert zugeordnet. Es wurden beiden Motoren jeweils die gleichen Motorwerte übergeben. Bei Werten von über 40 ist keine stabile Geradeausfahrt gegeben. Der Roboter bewegt sich dann auf einer Kreisbahn im Uhrzeigersinn.
In Abbildung 4.7 sieht man den Zusammenhang zwischen Motorwert und Geschwindigkeit. Er kann, unter Berücksichtigung von Messfehlern als linear angesehen werden.

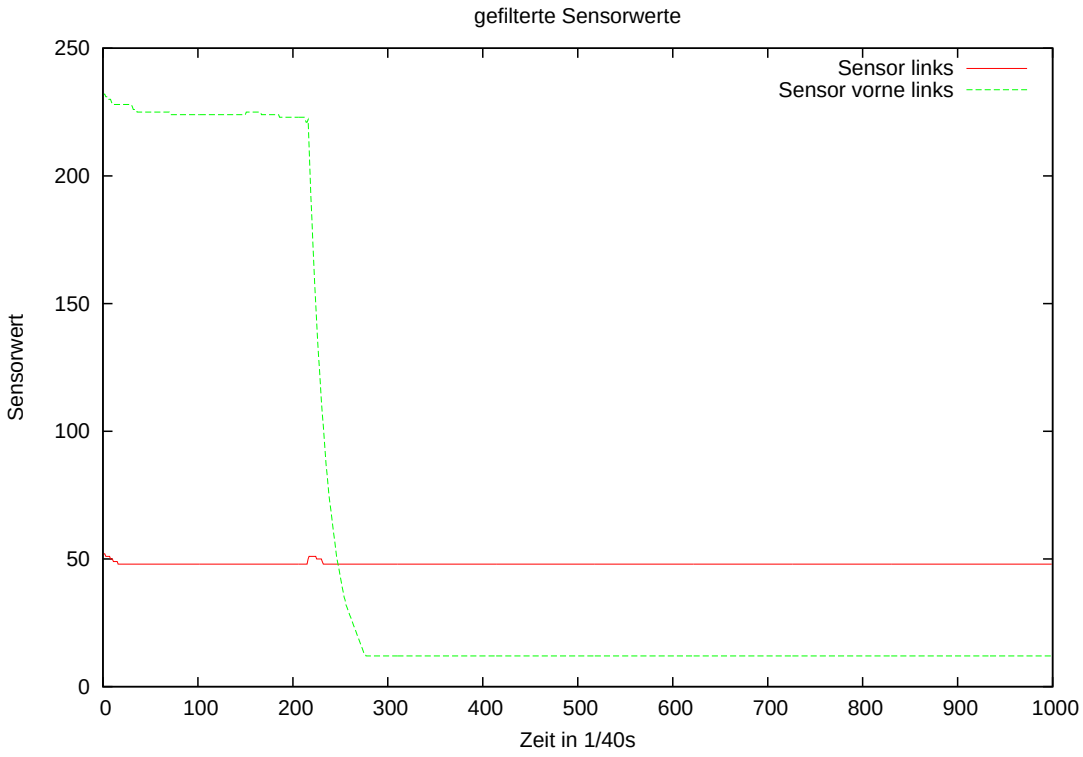

Abbildung 4.5: *Eine Zeitreihe von gefilterten Werten der Sensoren Nummer 0 (links), Nummer 2 (vorne links). Es steht ein Hindernis in 6 cm Entfernung direkt vor dem Roboter. Man sieht, dass das Rauschen beinahe komplett unterdrückt ist. Allerdings sieht man ab* $220/40$ *s den Nachteil der Filterung. Ab hier wurde das Hindernis ohne Zeitverzögerung entfernt. Die gefilterten Sensorwerte brauchen ein wenig Zeit, sich auf den neuen korrekten Wert einzustellen. .*

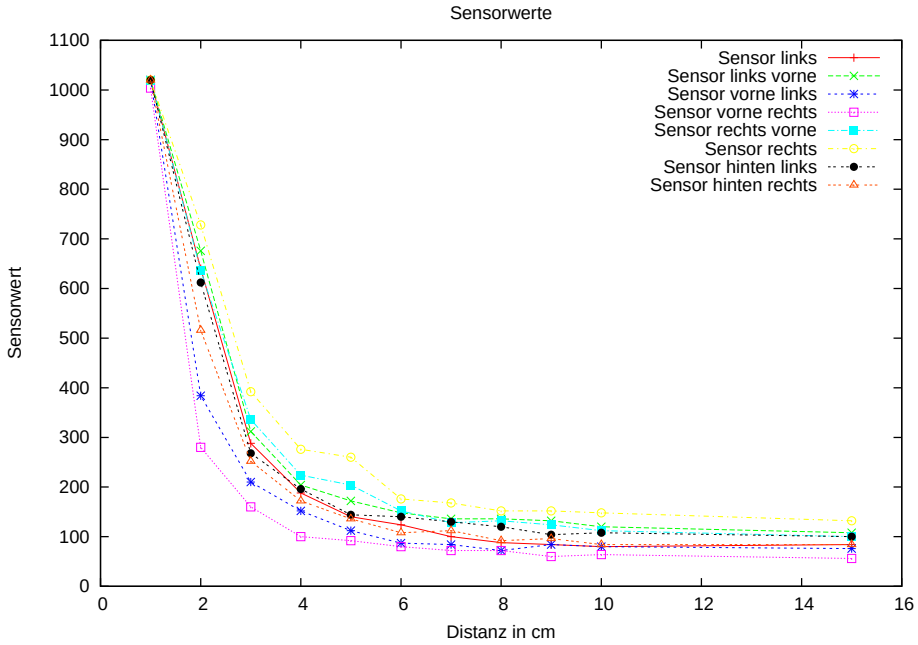

Abbildung 4.6: *Kennlinien der 8 Sensoren des Khepera II. Das Hindernis ist eine weiße Wand, senkrecht zur jeweiligen Ausbreitungsrichtung des Infrarotstrahls.*

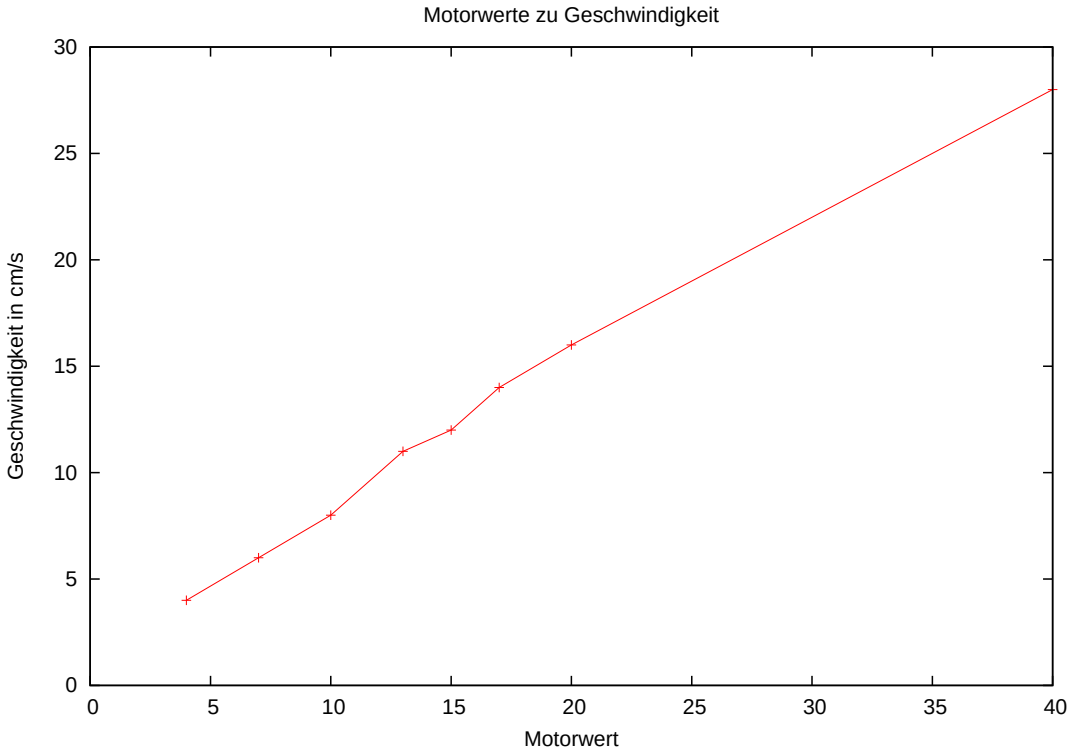

Abbildung 4.7: *Hier wird die Abhängigkeit von Motorwerten zur Geschwindigkeit gezeigt. Der Zusammenhang kann unter Berücksichtigung von Messfehlern als linear angesehen werden. Werte über 40 führten zu keiner Geradeausfahrt mehr.*

4.3 Der Pledge-Algorithmus auf dem Khepera II

Zu Beginn dieser Diplomarbeit wurde der Pledge-Algorithmus auf dem Khepera II Roboter implementiert. Als Beispiel für die Programmierung der Khepera II Plattform und weil dieser Pledge-Algorithmus die Grundlage für den Einbahnstraßen-Pledge ist, wird das Programm hier kurz dargestellt (der Source-Code findet sich im Anhang A).

4.3.1 Die Programmteile

Der Pledge-Algorithmus besteht in dieser Implementierung aus vier verschiedenen Prozessen, die sich gegenseitig aufrufen oder schlafen schicken.

Sensorabfrage

Die Sensorabfrage ist ein Prozess der im Schnitt alle 25 ms aufgerufen wird. Hier werden die aktuellen Sensoren abgefragt und in einem Array den anderen Prozessen zur Verfügung gestellt. Eine Filterung der Sensorwerte hat sich in Experimenten als nicht notwendig erwiesen. Dieser Prozess läuft ständig und kann nicht schlafen gelegt werden.

Winkelodometrie

Die Winkelodometrie merkt sich den Gesamtdrehwinkel, den der Roboter bis zu diesem Zeitpunkt gemacht hat. Dies wird durch die Formel aus der Odometrie (siehe Kapitel 2) realisiert. Durch das Vergleichen der Werte des Motor-Encoder von der letzten und der aktuellen Messung, kann der Winkel, der seitdem gefahren wurde, berechnet werden. Diese Winkeländerung wird auf den Winkelzähler addiert (diese Änderung kann negativ sein). Diese Berechnung wird alle 100 ms durchgeführt. Wenn nun der Winkelzähler größer Null plus einem (kleinen) Epsilon ist, wird der Hindernisvermeider angestoßen und der Wall Follower schlafen gelegt. Die Grenze „Null plus Epsilon“ wurde eingebaut, um es nicht notwendig zu machen, dass die Aktion des Winkelzählers davon abhängig ist, in welchem Steuerungsmodus das Programm sich gerade befindet.
Der Winkelzähler läuft die ganze Zeit und kann nicht schlafen gelegt werden.

Hindernisvermeider

Der Pledge-Algorithmus hat zwei verschiedene Modi: einen Wall-Follower und eine freie Bewegung. In diesem Prozess wird die freie Bewegung gesteuert. Der Prozess wird angestoßen, wenn der Winkelzähler den Wert Null erreicht.

Der Roboter bewegt sich so lange geradeaus nach vorne, bis einer der nach vorne gerichteten Sensoren ein Hindernis registriert. Dies geschieht, sobald der Wert eines nach vorne gerichteten Sensors einen bestimmten Grenzwert überschreitet. Die hierfür abgefragten Sensoren sind die Sensoren links-vorne (Sensor Nr. 1), rechts-vorne (Sensor Nr. 4), vorne-links (Sensor Nr. 2) und vorne-rechts (Sensor Nr. 3). Da die Sensoren Nummer 1 und Nummer 4 etwa in 45 Grad Winkel zur Fahrtrichtung gestellt sind, ist der Grenzwert, ab dem ein Hindernis als solches erkannt wird, höher gesetzt, als bei den beiden Frontsensoren Nummer 2 und 3. So wird eine Ecke, an der der Roboter noch problemlos vorbeifahren könnte, nicht als Hindernis erkannt.
Wenn ein Hindernis erreicht ist, wird der Wall-Follower Prozess angestoßen und der Hindernisvermeider schlafen gelegt.

Wall-Follower

Der Wall-Follower ist der zweite Modus der Pledge-Steuerung. Er wird aufgerufen, sobald der Hindernisvermeider ein Hindernis entdeckt hat.
Der Wall-Follower soll den Roboter so steuern, dass der Roboter die Hinderniswand immer auf der linken Seite hat. Dies wird dadurch realisiert, dass der Sensorwert des Sensors Nummer 0 zeitlich möglichst konstant gehalten wird. So wird ein gleichmäßiger Abstand zum Hindernis erreicht. Es wird also der Wert vom letzten Zeitschritt mit dem aktuellen Wert verglichen. Ist der Unterschied kleiner als Null, muss näher an die Wand gefahren werden. Ist der Unterschied größer Null, muss dem entsprechend von der Wand weggefahren werden. Zusätzlich gibt es noch einen Grenzwert, bei dem der Roboter stärker gesteuert wird. Dies ist nötig, da z.B. eine Ecke vom Roboter eine starke Drehung erfordert, damit die Wand nicht „verloren“ geht.
Zuletzt gibt es noch eine Abfrage, ob sich ein Hindernis direkt vor dem Roboter befindet. Hierfür wir der Sensor Nummer 2 benutzt. Registriert dieser ein Hindernis in unmittelbarer Nähe, dreht sich der Roboter im Uhrzeigersinn auf der Stelle, bis dieses Hindernis die Wand auf der linken Seite geworden ist, der man weiter folgen kann.
Der Wall-Follower wird vom Hindernisvermeider angestoßen und vom Winkelzähler schlafen gelegt.

Abhängigkeiten

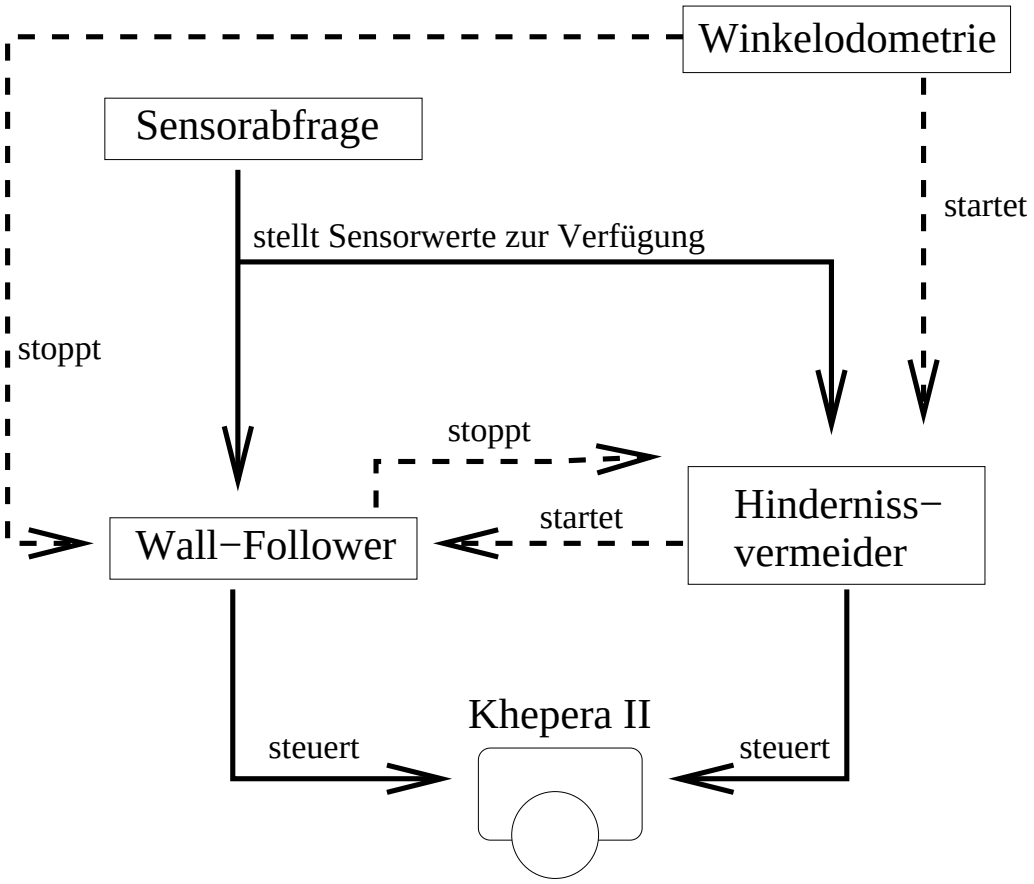

Abbildung 4.8: *Die Struktur und Abhängigkeiten des implementierten Pledge-Algorithmus auf der Khepera II Plattform*

4.4 Implementierung des Backtracking-Pledge

4.4.1 Erweiterte Fähigkeiten des Roboters

Wie im Kapitel 3 beschrieben, wird für den Backtracking-Pledge vom Roboter eine zusätzliche Eigenschaft gefordert: die Unterscheidung von Einbahnstraßen.
Verschiedene Möglichkeiten diese Fähigkeit zu implementieren sind denkbar, je nach dem, welche Sensorik der Roboter besitzt.
Beschränkt man sich jedoch auf die Möglichkeiten, die die Distanzsensoren des Kheperas bieten, so bedeutet dies eine Einschränkung der Labyrinthe, die gebaut werden könnten. Bedingt durch die relativ geringe Sichtweite (8 cm) wären breitere Einbahnstraßen nicht möglich. Zusätzlich ist die Erkennungssicherheit durch die Distanzsensoren (z.B. bei ein Implementierung über „Rippel“, über Vertiefungen die einen Binärcode übermitteln) sehr gering.
Deswegen wurden die Sensoren des Khepera durch einen Kamera-Turm erweitert. Der Kamera Turm ist die *CMUCam2* (siehe Abb. 4.9).
Die CMUCam2 ist eine Matrix-Kamera mit 176 mal 255 Bildpunkten. Die Kamera ist ein Farbkamera. Das Besondere an der CMUCam2 ist ein Vorverarbeitungschip direkt auf dem Turm, der eine Bildvorverarbeitung macht und die Ergebnisse an den Khepera übermittelt.

Abbildung 4.9: *Der Turm mit der CMUCam2 auf dem Khepera II.*

Standardmäßig kann dieser Chip drei verschiedenen Aufgaben übernehmen:

- Histogramm: Die Kamera kann ein Farbhistogramm über das gesamte Bild liefern. Sie gibt für jeden Farbkanal die Anzahl der Pixel im Bild an, die einen bestimmten Helligkeitsbereich in dieser Farbe haben. Die Werte von 0 - 255 werden auf 28 gleich große Bereiche diskretisiert.

- Bewegungsdetektion: Das Bild wird in 16 Bereiche eingeteilt. Durch den Vergleich von zwei aufeinanderfolgenden Bildern wird bestimmt, in welchen dieser Bereiche eine Bewegung stattgefunden hat. Sinnvoll ist diese Möglichkeit nur, wenn sich der Roboter nicht bewegt.

- Farbdetektion: Die Kamera sucht im Bild nach einem Bereich, der einen vorgegebenen Farbbereich besitzt. Sie gibt als Ergebnis aus:

  – X und Y Koordinate des Masseschwerpunktes des Farbbereiches (xmas und ymas).

  – Die Koordinaten der linken oberen Ecke und der rechten unteren Ecke des gefundenen Objektes, mit der gesuchten Farbe (x1,y1 und x2,y2).

  – Die Anzahl aller Pixel im gesuchten Farbbereich (pix). Maximalwert ist 255.

  – Ein „Vertrauen“ für das gefundene Objekt (conf). Der Wert errechnet sich aus pix/area * 256. Area ist die Anzahl von Pixel, die das umschließende Rechteck enthält.

Den großen Vorteil, den die Vorverarbeitung der Bilddaten liefert, kann leicht an einem Zahlenbeispiel gezeigt werden. Um ein komplettes Bild in den Speicher des Kheperas zu übertragen, braucht das KNET Protokoll etwa 10 Sekunden. Die Daten der Farbverfolgung können aber auf der Kamera selber mit etwa 17 Bildern pro Sekunde berechnet werden. Die Übertragung der 8 Werte (xmas,ymas,x1,y1,x2,y2,pix,conf) an den Khepera fällt zeitlich nicht ins Gewicht.

4.4.2 Die Programmteile

Im Gegensatz zum Pledge-Algorithmus, ist der Backtracking-Pledge nicht über verschiedene paralell laufende Prozesse realisiert worden. Dieses hat den Grund, das verschiedene Sensorsysteme (Distanzsensoren und Kamera) zusammen ausgewertet werden müssen. Wenn für jedes Sensorsystem ein eigener Prozess benutzt worden wäre, ist die konsitenz der Daten nihct gewährleistet, bzw. müsste Aufwendig sicher gestellt werden. Einzig die Odometrie ist ein eigener Prozess geblieben.
Der Steuerung des Khepera II ist über eine Finite State Machine (vgl. Abbildung 4.10). Die beiden gestarteten Prozesse sind:

Odometrie

Die Odometrieberechnung ist im Vergleich zum Pledge-Algorithmus im Einbahnstraßen-Pledge Typ2 unverändert. Allerdings erfüllt die Odometrie eine weiter Funktion. Sie überwacht nicht nur, wann der Winkelzähler wieder den Wert Null erreicht hat, sie stellt auch sicher, das der Roboter wieder in Startrichtung ausgerichtet ist, wenn die Gerade-aus BEwegung stattfindet. Dies ist notwendig, da bei der Steuerung gewisse „Todzeiten“ entstehen können, in denen der Roboter sich dennoch weiter bewegt. Diese „Todzeiten“ entstehen bei der Verarbeitung der Kameradaten. Um zu verhindern, dash hiedurch ein Steurungfehler entsteht, überprüft der Odometrie-Prozess jederzeit den aktuellen Winkel. Ist dieser größer als Null, so wird der Roboter so zurück gedreht, dass der WInkelzähler wiede auf Null steht.

Robotersteuerung

Die Robotersteuerung besteht aus der Sensorenauswertung und er eigentlichen Steuerung, die auf einer Finite State Machine beruht. Ja nach den Sensordaten wird von einem Zustand in einen anderen übergegangen

Sensorik und Bildauswertung

Der Programmteil Sensorik stellt die Werte der Sensoren der Robotersteuerung zu Verfügung. Allerdings muss hier wesentlich mehr geleistet werden als bei dem Pledge-Algorithmus, da die Sensorik wesentlich komplexer geworden ist. Der Roboter ist nun mit der Kamera ausgestattet. Die Interpretation der Einbahnstraßen wird in der Sensorik mitgeliefert.
Einbahnstraßen sind in der Spielwelt durch 2 farbige Streifen gekennzeichnet. Hierbei ist einer der Streifen Rot und der andere hat eine eindeutige, andere Farbe. Der rote Streifen ist die Markierung für die verbotene Seite. Trifft der Roboter als erstes auf diese rote Makierung bedeutet das für den Roboter, dass er eine Wand sieht (vergl. Zustände er Einbahnstraße in Kapitel 3). Der zweite Streifen ist der sogenannte *Identifikationsstreifen*. Seine Farbe signalisiert dem Roboter welche Einbahnstraße der Roboter gerade betritt. Dies bedeutet, diese Identifikationsfarbe muss für jede Einbahnstraße eindeutig sein. Trifft der Roboter also auf eine Farbe, die nicht Rot entspricht, so ist er von der erlaubten Seite an diese Einbahnstraße heran gekommen und dürfte durchfahren.

Die Zustände:

Zustand 0:

Dies ist der Startzustand für die Robotersteuerung. Hierbei ist auf keine Frabe getrof-

fen worden und der Roboter fährt Gerade-aus nach vorne. Der Winkelzähler ist gleich Null. Trifft der Roboter auf ein Hindernis geht die Robotersteuerung in den Zustand 1 über. Trifft Der Roboter auf einen roten Streifen, geht die Steuerung in den Zustand 2 über. Trifft der Roboter auf eine Farbe die nicht Rot ist, geht die Steuerung in den Zustand 4 über.

Zustand 1:
Der Roboter befindet sich im Wall-Follower Modus. Der Winkelzähler ist echt kleiner Null und der Roboter hat keine Farbe gefunden. Wird der Winkelzähler gleich Null, so geht die Steuerung in den Zustand Null über. Findet der Roboter einen roten Streifen, so geht die Steuerung in Zustand 3 über. Ist die Farbe eine ander als Rot (Identifikationsstreifen) so geht die Steuerung in Zustand 5 über.
Zustand 2:
Der Roboter steht an einer roten Makierung, die er nicht überfahren darf. Er fährt daran entlang. Ist die rote Makierung nicht mehr zu sehen, so geht die Stuerung in Zustand 0 über.

Zustand 3:
Der Roboter steht an einer roten Makierung, die er nicht überfahren darf. Er fährt daran entlang. Ist die rote Makierung nicht mehr zu sehen, so geht die Stuerung in Zustand 1 über.

Zustand 4:
Der Roboter fährt Gerade-aus. Der Winkelzähler ist gleich Null und er hat sich eine Identifikationsfarbe gemerkt (die Farbe, die veranlasst hat, dass die Steuerung von Zustand 0 nach Zustand 4 gewechselt ist). Trifft der Roboter jetzt auf ein Hindernis, so geht die Steuerung in Zustand 5 über. Findet der Roboter eine rote Makierung, so wird die Verhaltensliste benötigt. Diese wird, wie in Kapitel 3 beschrieben, aktualisiert. Soll die Einbahnstraße mit der aktuellen Identifikationsfarbe durchfahren werden, so tut der Roboter dies und wechselt in den Zustand 0. Soll die Einbahnstraße nicht durchfaren werden, so wechselt der Roboter in Zustand 2.

Zustand 5:
Der Roboter ist im Wall-Follower Modus. Der Winkelzähler ist kleiner Null und der Roboter sich eine Identifikationsfarbe gemerkt. Wird der Winkelzähler gleich Null, so geht die Steuerung in Zustand 4 über. Findet der Roboter eine rote Makierung, so wird ebenfalls wieder die Verhaltensliste benötigt. Diese wird, wie in Kapitel 3 beschrieben, aktualisiert. Soll die Einbahnstraße mit der aktuellen Identifikationsfarbe durchfahren werden, so tut der Roboter dies und wechselt in den Zustand 1. Soll die Einbahnstraße nicht durchfaren werden, so wechselt der Roboter in Zustand 3.

Verhaltensvorschriften Verwaltung
Die Verhaltenvorschriften Verwaltung bietet alle Funktionen, die für die Verwaltung der Verhaltensliste benötigt werden. Es ist eine einfach verkette Liste in der das Triplet {*Einbahnstraßen ID, Winkelzählerwert, „durchgehen/nicht durchgehen“*} gespeichert ist. Verschiedene Operationen können nun auf dieser Liste ausgeführt werden. Die Einfachsten sind: Einfügen eines Triplet am Ende der Liste, suche nach einem Triplet mit der ID als Schlüssel, einen Winkelzählerwert für eine bestimmte ID erhalten, Triplet anhand von ID löschen.
Zusätzlich übernimmt die „Verhaltensvorschriften Verwalten“ auch das Ändern des Verhaltens an einer Einbahnstraße, wenn ein Kreis gefunden wurde. Ein Kreis ist genau dann gefunden, wenn folgende Eigenschaften zutreffen:

- eine Einbahnstraße wurde getroffen, die schon in der Verhaltensliste gespeichert ist
- diese Einbahnstraße ist in der Verhaltensliste **nicht** Nachfolger der vorher besuchten Einbahnstraße

Sind diese beiden Voraussetzungen erfüllt, muss die Verhaltensweise am Ende der Liste geändert werden. Dies geschieht wie in Kapitel 3 beschrieben. Ist das Verhalten an der letzten Einbahnstraße gleich „nicht durchgehen“ so wird es auf „durchgehen“ gesetzt. Steht das Verhalten schon auf „durchgehen“, so wird das letzte Triplet gelöscht und die jetzige Verhaltensweise am Ende der Liste wird geändert. Dies wird so lange durchgeführt, bis entweder eine Verhalten von „nicht durchgehen“ auf „durchgehen“ gesetzt wurde, oder die Liste leer ist. Die Verhaltensliste ist eine globale Liste, so das jeder Prozess in dieser Liste nachhalten kann, ob eine Einbahnstraße durchfahren werden soll, oder nicht.
Die Verhaltensvorschriften Verwaltung besteht nur aus Operationen auf der Liste, so dass sie keinen eigenen Prozess darstellt. Demzufolge ist sie nur aktiv wenn in einem anderen Prozess hierauf zugegriffen wird.

Umgehung der Rekursion

Der Backtracking-Pledge ist im Kapitel 3 als rekursiver Algorithmus beschrieben. Zur Erinnerung, der Algorithmus ruft sich rekursiv auf, sobald dem Roboter bei der Änderung von Verhaltensweisen „der Boden unter den Füßen weggezogen wurde“, d.h. bei Ankunft an der Einbahnstraße war diese noch in der Verhaltensliste. Es wurde ein Kreis festgestellt und die Verhaltensweise geändert. Nun befindet sich die Einbahnstraße nicht mehr in der Verhaltensliste. Der Backtracking-Pledge fängt also mit dieser Einbahnstraße erneut an, bis entweder der Ausgang gefunden wurde, oder eine aus den Rekursionsebenen vorher bekannte Einbahnstraße gefunden wurde.

Da der rekursive Aufruf des Einbahnstraße-Pledge Typ2 relativ viele Ressourcen verbraucht und dies bei einem autonomen Lauf des Khepera II kritisch werden könnte, wurde auf die Rekursion verzichtet. Dem Triplet, welches in der Verhaltensliste gespeichert wird, wird ein Wert mitgegeben, in welche Rekursionsstufe die Einbahnstraße gespeichert wurde. Stellt der Algorithmus fest, dass die Einbahnstraße, die der Roboter gerade erreicht hat, durch das Ändern der Verhaltensweise gelöscht wurde, wird der Rekursionszähler um einen erhöht und der Winkelzähler auf Null gesetzt. Wenn weitere Einbahnstraßen erreicht werden, so werden sie mit dem aktuellen Rekursionszähler in der Liste gespeichert, wenn sie noch nicht in der Liste sind. Ist eine Einbahnstraße auf die der Roboter mit höherem Rekursionszähler trifft, schon in der Liste gespeichert und zwar mit einem niedrigeren Rekursionszähler, so wird damit wie folgt verfahren:

- Rekursionzähler auf den niedrigen Wert setzen
- Alle Tripels, die einen höheren Wert haben als der Rekursionszähler, werden gelöscht
- Der Winkelzähler wird auf den gespeicherten Wert gesetzt
- Die Verhaltensweise am Ende der Liste wird **nicht** geändert

Dieses Vorgehen führt letztlich zu gleichem Verhalten des Algorithmus, spart allerdings auf einem realen System wie dem Khepera II Ressourcen.

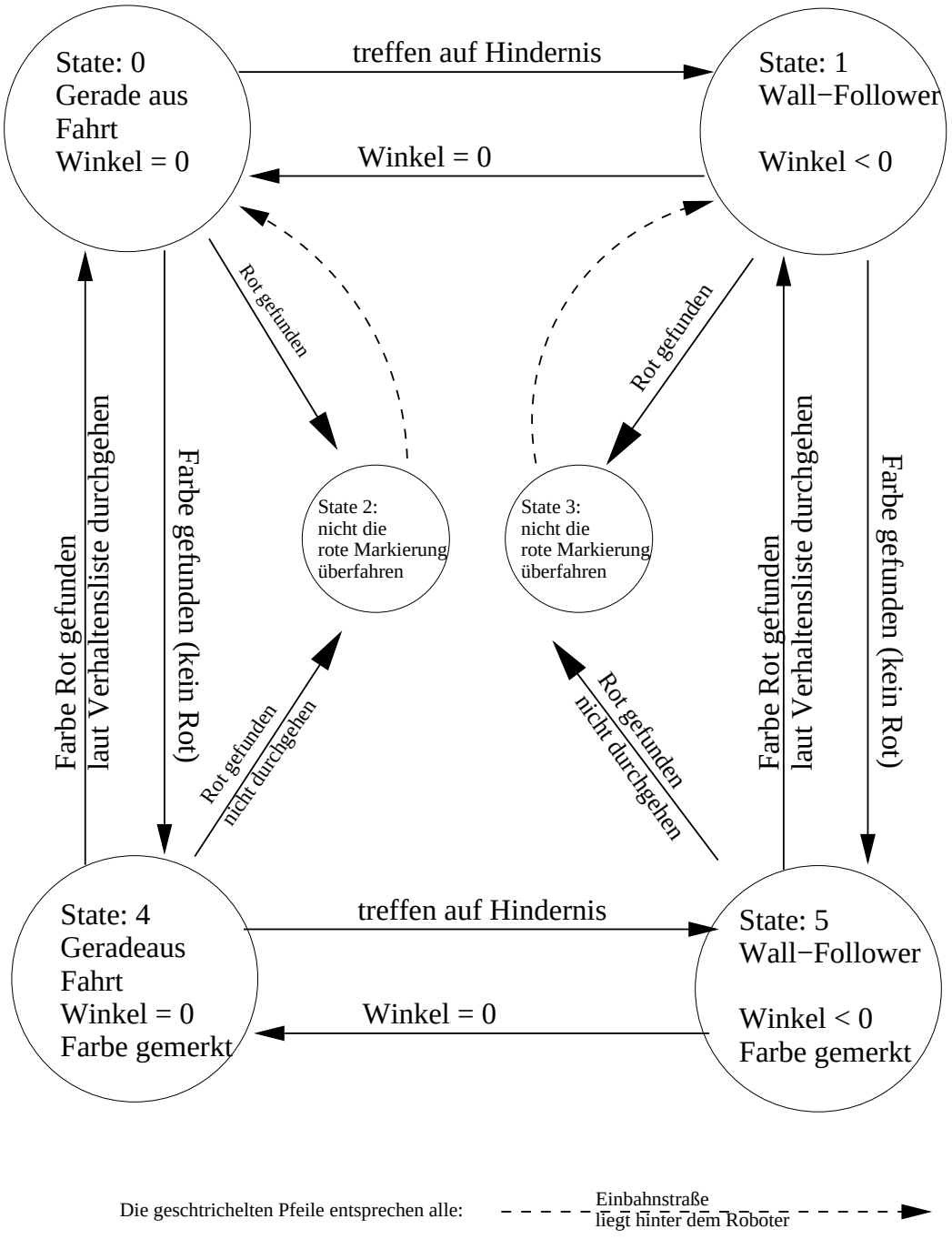

Abbildung 4.10: *Zustandsdiagramm des auf dem Khepera II implementierten Backtracking-Pledge.*

4.4.3 Probleme und Sonderfälle in der Praxis

Der Pledge-Algorithmus auf dem Khepera II Roboter hat sich als sehr robust und zuverlässig erwiesen. Bei Experimenten zeigte sich, dass auch bei vielen Drehungen (z.B. bei einem spiralförmigen Labyrinth) Fehler in der Odometrie nicht auffällig wurden. Da die Steuerung des Backtracking-Pledge auf genau diesem Pledge-Algorithmus aufgebaut ist, sind auch hier keine Probleme aufgetreten.
Um den Backtracking-Pledge auf einem Roboter zu implementieren, muss der Roboter einerseits überhaupt Einbahnstraßen erkennen, anderseits muss er auch unterschiedliche Einbahnstraßen voneinander unterscheiden können. Um dies zu realisieren, wurde dem Khepera II die CMUCam als zusätzliches Sensorsystem mitgegeben. Die Möglichkeit, Einbahnstraßen zu erkennen, wurde durch Farbmarkierung auf dem Boden realisiert. Dies bedeutet, dass der Roboter nach bestimmten Farben suchen muss. Hier zeigt sich ein großes Problem in der praktischen Umsetzung: Nach einer bestimmten Farbe zu suchen erweist sich als schwierig. So reichen schon unterschiedliche Lichtbedingungen, um einen völlig verschiedenen Farbeindruck zu bekommen. Der Backtracking-Pledge auf dem Khepera II kann in dieser Implementierung nur bei kontrollierten Lichtverhältnissen ausgeführt werden. Kontrolliert bedeutet hier, ein gleichmäßiges Neonlicht ohne Einstrahlung von Sonnenlicht.
Da die Kamera nach vorne gerichtet ist, musste ihre Blickrichtung mt Hilfe eines Spiegels umgelenkt werden (vgl. Abb. 4.11). Dieser Spiegel aber würde gegen die Wände stoßen. So musste die Höher der Hindernisse in der Spielwelt so angepasst werden, dass sowohl die Distanzsensoren die Hindernisse noch wahrnehmen, der Spiegel aber über die Hindernisse hinweg geführt werden kann.

Ebenso ist die Farbverfolgung nicht dafür geeignet, nach den Farbcodierungen für die Einbahnstraßen zu suchen. Es zeigte sich im praktischen Versuch, dass hierfür zu viele Anfragen pro Sekunde an die Kamera gestellt werden müssen um noch eine akzeptable Steuerungsgeschwindigkeit zu erhalten. Hierbei passiert es häufig, dass die Kamera nach kurzer Zeit in einen Fehlerzustand gelangte und nur durch einen Hardware Reset wieder in einen funktionstüchtigen Zustande gebracht werden kann. So wurde die Erkennung von Farben über eine inverse Farbsuche gestaltet. Die Lösung besteht darin, dass nach allen Farben gesucht wird, welche nicht der Bodenfarbe entsprechen (dies kann über eine einzige Abfrage realisiert werden, da die Kamera eine inverse Farbsuche erlaubt). Hierbei stellte es sich als sinnvoll heraus, das Labyrinth weiß zu streichen (vgl. Abb. 4.12) Ist so eine Farbe gefunden, hält der Roboter an und untersucht nacheienader die ihm bekannten Farben, ob diese in dem aktuellen Bild vorkommen. Auf diese Weise entsteht eine kurze „Denkpause“ wenn der Roboter an eine Farbmarkierung gelangt.

Bei den praktischen Experimenten zeigte sich, dass Fehler in der Farberkennung unterschiedliche Auswirkungen haben können. So sind drei verschiedene Fehler denkbar:

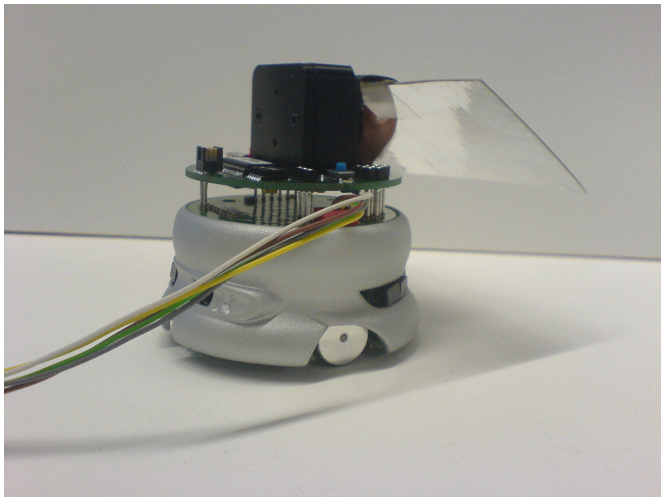

Abbildung 4.11: *Damit der Khepera II nur den Boden nach einer Einbahnstraße absucht, ist die Blickrichtung der Kamera mit einem Spiegel umgelenkt.*

- Erkennen keiner Farbe, obwohl eine Farbe vorhanden ist
- Erkennen einer Farbe, obwohl keine Farbe vorhanden ist („Phantomfarbe“)
- Erkennen einer falschen Farbe

Das Erkennen keiner Farbe, obwohl eine Farbe vorhanden ist, ist in den praktischen Versuchen nicht aufgetreten. Sollte dies allerdings vorkommen, ist anzunehmen, dass hierdurch der gesamte Algorithmus durcheinander gerät, bzw. Aktionen volführt die nicht Möglich sein sollten (Durchfahren einer Einbahnstraße von der verbotenen Seite). Das Problem der Phantomfarbe ist ein Problem, welches sich als reales Problem herrausgestellt hat. In wie weit dieses Problem den Algorithmus beeinflusst, hängt von der aktuellen Situaton ab. Anzumerken sei vorher, dass das Erkennen einer Phantomfarbe nicht an eine bestimmte Stelle gebunden ist, d. h. wird eine Stelle zwei Mal überfahren wird nicht zwangsläufig jedes Mal die Phantomfarbe erkannt. So erzeugt ein falsch Erkanntes Rot keine Pobleme, da der Roboter nur kurz denken, hier könnte nicht passiert werden. Nach einer kurzen Bewegung ist die Phantomfarbe wahrscheinlich nicht mehr vorhanden.
Es gibt drei unterschiedliche Situaion in denen ein Rauschen der Farberkennung in Form der Phantomfarbe unterschiedliche Auswirkung haben:

- die nächste reale Farbe ist ein Identifikationsstreifen

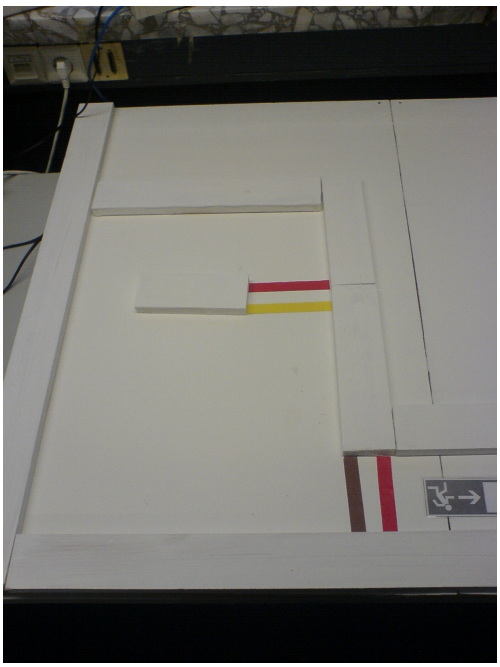

Abbildung 4.12: *Ein Beispiellabyrinth mit der modifizierten Testwelt. Die Hindernisse sind niedriger, so dass der Spiegel darüber ragen kann. Auch musste die Spielwelt weiß gestrichen werden, so dass die Farbe des Holzes nicht für eine falsche Farberkennung sorgen kann.*

- die nächste reale Farbe ist Rot, allerdings noch eine längere Strecke entfernt
- die nächste reale Farbe ist Rot, in unmittelbarer Nähe

Im ersten Fall erzeugt die Phantomfarbe keinerlei Fehlverhalten des Algorithmus. Die Phantomfarbe wird bei Erreichen des Identifikationsstreifens überschrieben und hat keinerlei Auswirkungen. Auch der zweite Fall hat keine Auswirkungen. Da bekannt ist, dass auf eine Farbe Rot folgen muss, damit es sich um eine Einbahnstraße handelt und zustätzlich bekannt ist, dass dieses Rot nahe bei der Identifikationsfarbe liegen muss; kann durch Vergleich der aktuellen Zeit (die Zeit, als die Farbe Rot gefunden wurde) mit dem Zeitpunkt zu dem die Identifikationsfarbe gefunden wurde, bestimmt werden, dass es sich hier um eine Phantomfarbe handelt. Der Roboter nimmt das Rot richtigerweise als verbotener Seite einer Einbahnstraße auf. Hier sieht man auch das Problem, dass sich durch eine Phantomfarbe in unmittelbarer Umgebung der roten Makierung ergibt. Dies kann von dem Roboter nicht von einer realen Einbahnstraße

unterschieden werden und der Roboter volführt evetuell eine verbotene Aktion, indem er die Einbahnstarße von der verbotenen Seite her überquert.
Das Erkennen einer falschen Farbe kann entweder dazu führen, dass der Roboter eine Einbahnstraße falsch oder gar nicht identifiert, oder das er eine illegale Aktion durchführt, in dem er eine Einbahnstraße von der verbotenen Seite durchfährt.

Kapitel 5

Diskussion und Ausblick

Die Aufgabenstellung für diese Diplomarbeit war vielschichtig. Zum Einen sollte ein Algorithmus entwickelt werden, der aus jedem beliebigen Labyrinth mit Einbahnstraßen herausfinden soll. An diesen Algorithmus war die Anforderung gestellt worden, dass er in seiner Komplexität dem Pledge-Algorithmus ähnlich ist. Dies heißt, dass vor allem die Speicheranforderung möglichst gering bleiben sollte.
Die zweite Aufgabe war die Umsetzung des Pledge-Algorithmus auf dem Khepera II Roboter. Für diese Aufgabe musste eine geeignete Umgebung geschaffen werden.
Zuletzt sollte der in dieser Arbeit entwickelte Algorithmus ebenfalls auf dem Khepera II implementiert werden.

Bei der Entwicklung des Algorithmus zum Entkommen aus unbekannten Labyrinthen mit Einbahnstraßen wurde in dieser Arbeit der Pledge-Algorithmus als Grundlage genommen und es hat sich gezeigt, dass sich der Pledge-Algorithmus durch die Entwickelung von Verhaltensweisen zu einem Algorithmus modifizieren ließ, der Labyrinthe mit Einbahnstraße lösen kann. Da sich Verhaltensweisen auf zwei verschieden Arten beschreiben lassen, einmal als lokale Verhaltensweise und einmal als Globale, wurde eine Klassifizierung von Einbahnstraßen-Pledge Algorithmen vorgenommen. Die Algorithmen wurden unterschieden in solche, die nach globalen Verhaltensweisen suchen (*Einbahnstraßen-Pledge Typ1*) und die, die nach globalen Verhaltensweisen suchen (*Einbahnstraßen-Pledge Typ2*). Als erstes wurde ein Algorithmus vorgestellt der zum ersten Typ gehört. Dieser Algorithmus (Steuerwort-Einbahnstraßen-Pledge) führt, wie in dieser Diplomarbeit bewiesen, aus jeden Labyrinth mit Einbahnstraßen heraus. Er ist aber auf Grund seiner Laufzeit nicht geeignet, auf einem realen Roboter umgesetzt zu werden. Zusätzlich erfüllt der Steuerwort-Einbahnstraßen-Pledge streng genommen nicht die Vorgaben, die zu Beginn der Diplomarbeit an den Algorithmus gestellt wurden. Der Steuerwort-Einbahnstraßen-Pledge erzeugt ein Steuerwort, mit dessen Hilfe der Roboter durch das Labyrinth navigiert. Die Länge dieses Steuerwortes liegt allerdings in $O(N^2)$ wobei N die Anzahl der Einbahnstraßen ist. Da dieses

Wort gespeichert werden muss, hat der Steuerwort-Einbahnstraßen-Pledge einen ebenso großen Speicherbedarf[1]. Aus diesen Gründen wurde nach anderen Algorithmen gesucht. Durch die Erweiterung des Robotermodells um die Fähigkeit, Einbahnstraßen voneinander unterscheiden zu können, war die Möglichkeit gegeben, nach einer globalen Verhaltensweise zu suchen.

In dieser Diplomarbeit sind drei verschieden Algorithmen vorgestellt worden, die jeweils nach einer globalen Verhaltensweise suchen. Jeder dieser Algorithmen benutzt die Informationen die durch die Unterscheidbarkeit der Einbahnstraßen gegeben ist, anders aus. Die drei Algorithmen sind der Binärzahl-Pledge, der Gebiets-Pledge und der Backtracking-Pledge. Diese Algorithmus erfüllt sowohl die Bedingung eines geringen Speicherbedarfs als auch die Umsetzbarkeit auf einen realen Roboter.

Bei der Umsetzung des Backtracking-Pledge auf den Khepera II muss auch dieser um die Möglichkeit erweitert werden, Einbahnstraßen zu unterscheiden. Hierzu wurde eine Kamera benutzt, die durch Farbcodierungen diese Einbahnstraßen sowohl wahrnehmen, wie auch eindeutig zuordnen konnte. Hierbei zeigte sich, dass die Farbcodierung, obwohl sehr schlüssig, doch sehr aufwendig ist. Der Farbeindruck über eine Kamera ist doch sehr unterschiedlich zum Farbempfinden des menschlichen Gehirns.

In dieser Diplomarbeit ist der Pledge-Algorithmus um die Fähigkeit erweitert worden, Labyrinthe mit Einbahnstraßen zu lösen. Eine weitere mögliche Fragestellung ist die, inwieweit der Pledge-Algorithmus um weitere Fähigkeiten erweitert werden könnte, um andere Labyrintharten zu lösen. So sind z.B. Labyrinthe mit beweglichen Hindernissen oder Labyrinthe, die mit verstreichender Zeit ihr Aussehen verändern, denkbar.

Ebenso zeigt diese Diplomarbeit Eigenschaften von Labyrinthen mit Einbahnstraßen auf. Mit Hilfe der entwickelten Konstrukte wie z.B. dem Begriff des Gebietes und deren Eigenschaften lässt sich das Verhalten von anderen Navigationsalgorithmen in Labyrinthen mit Einbahnstraßen einfacher analysieren. Ebenso wird wahrscheinlich eine Modifikation (wenn sie denn vorgenommen werden muss) der verschiedenen BUG-Algorithmen durch die erarbeiteten Eigenschaften und Konstrukte vereinfacht. Verschiedene Eigenschaften der Labyrinthe mit Einbahnstraßen könnten auch genutzt werden, um einen Navigationsalgorithmus zu entwickeln, der den Ausgang im Labyrinth findet, ohne auf dem Pledge-Algorithmus aufzubauen. Solch ein Algorithmus kann einen höheren Ressourcenbedarf haben, aber andere Bedingungen minimieren. So kann z.B. die Anzahl der Durchfahrten durch eine Einbahnstraße minimiert werden. Der vorgestellte Backtracking-Pledge beispielsweise, kann je nach Aussehen des

[1] Allerdings ist der Vorfaktor sehr gering, da für jeden Buchstaben des Steuerwortes nur 2 Bit gespeichert werden müssen

Labyrinthes, eine bestimmte Einbahnstraße beliebig (aber endlich) oft durchfahren.

Zusätzlich wurde gezeigt, dass der Backtracking-Pledge auch praktisch auf einem Roboter mit relativ geringen Ressourcen eingesetzt werden kann. Für Roboter mit besseren Möglichkeiten (Speicher, Rechengeschwindigkeiten, Sensoren) wird es allerdings andere Möglichkeiten geben, die zwar einen deutliche höheren Speicherbedarf und wesentlich mehr Rechenzeit bedürfen, aber viele Mehrfach-Wege vermeiden.

Literaturverzeichnis

[Ad80] ABELSON, H. und A. DIESESSA: *Turtle Geometry*. MIT Press, Cambridge, MA, 1980.

[BBC+95] BUHMANN, JOACHIM M., WOLFRAM BURGARD, ARMIN B. CREMERS, DIETER FOX, THOMAS HOFMANN, FRANK E. SCHNEIDER, JIANNIS STRIKOS und SEBASTIAN THRUN: *The Mobile Robot RHINO*. AI Magazine, 16(2):31–38, 1995.

[BEF96] BORENSTEIN, J., H. R. EVERETT und L. FENG: *Where am I*, 1996.

[Fra00] FRANZI, E.: *Khepera bus and turret specifications*, 2000.

[GKR99] GOERKE, N., F. KINTZLER und A. RAABE: *Golf Playing Khepera*, 1999.

[Hem93] HEMMERLING, ARMIN: *Navigation Without Perception of Coordinates and Distances*. Technischer Bericht TR-93-018, Berkeley, CA, 1993.

[Kam05] KAMPHANS, TOM: *Models and Algorithms for Online Exploration and Search*. Dissertation, University of Bonn, 2005.

[Kle05] KLEIN, ROLF: *Algorithmische Geometrie*. Springer, 2005.

[KT02a] K-TEAM: *Khepera*^{2} *Programming Manual*, 2002.

[KT02b] K-TEAM: *Khepera*^{2} *User Manual*, 2002.

[LDW91] LEONARD, J. und H. F. DURRANT-WHYTE: *Mobile Robot Localization by Tracking Geometric Beacons.* In: *IEEE Transactions on Robotics and Automat*, Band 7, Seiten 376–382, 1991.

[LMR99] LÖFFLER, A., F. MONDADA und U. RÜCKERT: *Experiments with the Mini-Robot Khepera*. 1999. Proceedings of the 1st International Khepera Workshop, Heinz Nixdorf Institut, Paderborn, Germany.

[LS86] LUMELSKY, V. J. und A. A. STEPANOV: *Dynamic path Planning for a Mobile Automaton with Limited Information on the Enviroment.* In: *IEEE transactions on Automatic control*, Seiten 1058–1063, 1986.

[MFI94] MONDADA, FRANCESCO, EDOARDO FRANZI und PAOLO IENNE: *Mobile Robot Miniaturisation: A Tool for Investigation in Control Algorithms*. In: *Experimental Robotics III*, Seiten 501–513, Kyoto, 1994. Springer-Verlag.

[MM95] MARTINOLI, A. und F. MONDADA: *Collective and Cooperative Group Behaviours: Biologically Inspired Experiments in Robotics*, 1995.

[RKSI93] RAO, N., S. KARETI, W. SHI und S. IYENAGAR: *Robot Navigation in Unknown Terrains: Introductory Survey of Non-Heuristic Algorithms*, 1993.

[Sha52] SHANNON, CLAUDE E.: *Presentation of a Maze Solving Machine*. In: VON FOERSTER, H., M. MEAD und H. L. TEUBER (Herausgeber): *Cybernetics: Circular, Causal and Feedback Mechanisms in Biological and Social Systems, Transactions Eighth Conference, 1951*, Seiten 169–181, New York, 1952. Josiah Macy Jr. Foundation.

[SM92] SANKARANARAYANAN, A. und I. MASUDA: *A new algorithm for robot curvefollowing amidst unknown obstacles, and a generalization of maze-searching.* In: *Proceedings of 1992 IEEE International Conference on Robotics and Automation*, Seiten 2487–2494, 1992.

[SV90a] SANKARANARAYANAN, A. und M. VIDYASAGAR: *A new path planning algorithm for a point object amidst unknown obstacles in a plane.* In: *Proceedings of IEEE Conference on Robotics and Automation*, Seiten 1930–1936, 1990.

[SV90b] SANKARANARAYANAN, A. und M. VIDYASAGAR: *Path planning for moving a point object amidst unknown obstacles in a plane: A new algorithm and a general theory for algorithm developments.* In: *Proceedings of IEEE Conference on Decision and Control*, Seiten 1111–1119, 1990.

[SV90c] SANKARANARAYANAN, A. und M. VIDYASAGAR: *Path planning for moving a point object amidst unknown obstacles in a plane: The universal lower bound on the worst case path lengths, and a classification of algorithms.* In: *Proceedings of IEEE Conference on Robotics and Automation*, Seiten 1734–1741, 1990.

Anhang A: Source-Code

```
/*
   ; Pledge Algorithem
   ; =======================

   ;-------------------------------------------
   ; Author:    Bernd Brueggemann      20/02/06
   ; Modifs:
   ;
   ; Project:   Diplomarbeit
   ; Goal:                  Implemnets the well
   ;                        known Pledge Algorithem
   ;                        on the Khepera II
   ;                        Platform
   ;                        This PRGM launches
   ;                        4 processes.
   ;             Process 0: Provides the programm
   ;                        with  sensor values
   ;             Process 1: does the odometrie
   ;                        and fire an event if
   ;                        counter >0;
   ;             Process 2: does a walk forward.
   ;                        If an obstacle appears
   ;                        it starts process 2
   ;             Process 3: does a wall follower
   ;                        until Process 1 fires
   ;                        event
   ;
   ;
   ;
   ;-------------------------------------------
*/

#include         <sys/kos.h>
#include         <stdlib.h>
#include         <stdio.h>

static int32 vIDProcess[4];

#define ALPHA 0.95
#define EPSILON 0.5
#define RADABST 0.10
#define TICK_IN_M 0.001
#define SPEED 10.0
#define FRONT_OBST 300
#define SIDE_OBST 400
#define UPPER_BOUND 500
#define LOWER_BOUND 300
#define NICHTS 90
#define DRIFT 5

uint32 act_sensors[8];
int32 sens[8];


double alpha;

int32 wallfollow;

/*
 * Process 0
 * ^^^^^^^^^
 *
 * Provides other processes with filtered IR Values
 *
 *
 */

static void
process_0 ()
 {
 int32 i;
         for(;;)
         {

                  for(i = 0;i < 8; i++)
                 {
                         act_sensors[i] = sens_get_reflected_value(i);
                 }



                 /** Suspend for 25 ms**/

                 tim_suspend_task(25);


         }
}

/*
 * Process 1
 * ^^^^^^^^^
 *
 * Counts turns a generate event if counter > 0 + epsilon
 *
 *
 */

static void
process_1 ()
{

int32 r,l;
int32 old_r,old_l;

old_r = 0;
old_l = 0;

  for (;;)
    {
           tim_lock();
         r = mot_get_position(0) - old_r; ;
         l = mot_get_position(1) - old_l ;
         old_r = r + old_r;
         old_l = l + old_l;
         tim_unlock();

         tim_lock();
         alpha = alpha+(((l*TICK_IN_M)-(r*TICK_IN_M))/(RADABST));
         tim_unlock();

         if ((alpha  > 0) && (wallfollow == 1))
          {
                 tim_generate_event();
                 wallfollow = 0;
                 alpha = 0;
          }

               tim_suspend_task (100);
      }
}

/*
 * Process 2
```

```
 * ^^^^^^^^^
 *
 * Does walk foward until obstacle found
 *
 */

static void
process_2 ()
{

  for (;;)
  {
   if (wallfollow == 1 )
           {
        tim_wait_event(vIDProcess[1]);
        }
   var_on_led(1);
   var_off_led(0);
   mot_new_speed_2m(SPEED,SPEED);

   if ((act_sensors[2] > FRONT_OBST)
                || (act_sensors[3] > FRONT_OBST)
                || (act_sensors[1] > SIDE_OBST)
                || (act_sensors[4] > SIDE_OBST))
           {
        /** obstacle found**/

        tim_generate_event();
        mot_new_speed_2m(0,0);
        wallfollow = 1;
        }

   tim_suspend_task(150);

   }
}

/*
 * Process 3
 * ^^^^^^^^^
 *
 * Wall-follower
 *
 */

static void
process_3 ()
{

int32 last_left = 0;
int32 diff;
int32 speed_left = SPEED;
int32 speed_right = SPEED;
  for (;;)
 {
        if (wallfollow == 0)
                tim_wait_event(vIDProcess[2]);

        var_on_led(0);
        var_off_led(1);

        if (last_left == 0)
                last_left = act_sensors[0];

        tim_lock();
        diff = act_sensors[0] - last_left;
        last_left = act_sensors[0];
         tim_unlock();

        speed_left = SPEED;
        speed_right = SPEED;

        if (diff > 0)
        {
                speed_left = SPEED - 2;
                speed_right = SPEED;
        }

        if (diff < 0)
        {
                speed_right = SPEED - 2;
                speed_left = SPEED;
        }

        if (act_sensors[0] < 200)
        {
                speed_left = SPEED - 10;
                speed_right = SPEED;
        }

        if (act_sensors[0] > 300)
        {
                speed_right = SPEED -10;
                speed_left = SPEED;
        }

        if (act_sensors[2] > 300)
        {
                speed_right = SPEED * -1;
                speed_left = SPEED;
        }

        mot_new_speed_2m(speed_right,speed_left);

        tim_suspend_task (50);
 }
}
/*
 * MAIN
 * ====
 *
 * - Initialise the used managers:
 *      tim & bios are initialised at the start-up.
 * - Launch all the processes.
 * - Kill the "main": At this moment only the launched
 *      processes are executed.
 *
 */

int
main (void)
{
  int32 status;
  int32 i;

  for (i = 0; i < 8;i++)
  {
          act_sensors[i] = 0;
  }
  wallfollow = 0;

  tim_reset();
  sens_reset();
  mot_reset();

  status = install_task ("Process name 0", 800, process_0);
  if (status == -1)
    exit (0);
  vIDProcess[0] = (uint32) status;

  status = install_task ("Process name 1", 800, process_1);
  if (status == -1)
    exit (0);
  vIDProcess[1] = (uint32) status;

  status = install_task ("Process name 2", 800, process_2);
  if (status == -1)
    exit (0);
  vIDProcess[2] = (uint32) status;

  status = install_task ("Process name 3", 800, process_3);
  if (status == -1)
    exit (0);
  vIDProcess[3] = (uint32) status;


  return 0;
}
```