Pfadfindungssystem für mobile Roboter

Diplomarbeit
an der
Technischen Universität Graz

von

Norbert Nock
Matr.Nr. 9230235

Institut für Grundlagen der Informationsverbreitung (IGI)
technische Universität Graz
A-8010 Graz, Österreich

Graz September 2005

o. UNIV. Prof. Dr. Wolfgang Maass

ich versichere, diese Arbeit selbständig verfasst, andere als die angegeben Quellen und Hilfsmittel nicht benutzt zu haben und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Norbert Nock
Graz, September 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

Kurzfassung:

Für einen Atonomen Mobilen Roboter (AMR) wurde eine Simulationumgebung programmiert, die auf geometrischen Karten mit Linen als Hindernissen basiert. Zusätzlich wurde ein topologisches Netzwerk über die geometrischen Karten gelegt, um die Karten möglichst klein zu halten und die Suche der Pfade auf kleine Teilbereiche zu beschränken. Um einem Pfad innerhalb der geometrischen Karten der Simulationsumgebung zu finden, wird mit dem A* Algorithmus ein Netzwerk aus Ausweichknoten und zielgerichtet erstellten Kanten erstellt, das den Startpunkt S mit dem Zielpunkt G verbindet.

Abkürzungen:

Linux	Abkürzung für GNU/LINUX Beriebsystem
QT	Development Tool Kit für grafische Benutzeroberfläche von Tolltech
OS	Operating System
AMR	Autonomer Mobiler Roboter
m_B	Breite des AMR
m_r	$m_B/2$ (Radius des AMR)
n	Wegpunkte
w	Wegstrecken
P	Positionen bzw. Punkte in der Ebene
N	Menge der Wegpunke (Knoten des Wegenetzes)
W	Menge der Wege (Kanten des Wegenetzes)
MAP	Karte mit Hindernissen die die Simulationsumgebung beschreibt
WN	Wegenetz
r_B	Bubbleradius
B	Bubble
B_{MP}	Bubblemittelpunkt

1 Zielsetzung

Es soll eine 2D Simulationsumgebung für einen mobilen Roboter entstehen (programmiert werden). Dabei sollen Hindernisse durch einfache Linien (Strecken) im zweidimensionalen Raum dargestellt werden. Hindernisse können in der Umgebung an beliebigen Stellen eingefügt werden.

Innerhalb dieser Umgebung soll durch ein Pfadfindungssystem ein durch den Roboter befahrbarer Pfad entstehen. Für den Roboter gilt:

- der Roboter hat eine kreisrunde Grundfläche
- der Roboter kann sich in jede Richtung bewegen; er kann also ohne Wenderadius die Fahrtrichtung ändern.

2 Vergleich von verschieden Verfahren zur Darstellung der Simulationsumgebung und zur Pfadsuche

Prinzipiell wird unterschieden zwischen Freiraum- und Hindernisdarstellung zum Einen und Graphen- und Rasterdarstellung zum Andern.
Welche Darstellung für ein gegebenes Problem am besten ist, hängt von vielen Aspekten ab. Dazu muß man sich zuerst die verschiedenen Qualitätskriterien für die Pfadsuche ansehen, welche sich auch gegenseitig ausschließen können: z.B. Der kürzeste Weg ist nicht immer der schnellste und sicherste, denn viele Kurven oder eine bestimmte Bodenbeschaffenheit können Einfluss auf die Geschwindigkeit haben, mit der der AMR den Pfad abfahren kann.

Hier einige Kriterien für die Suche nach dem optimalen Pfad:

- kollisionsfreier Pfad: Der Pfad soll so geplant werden, daß er möglichst kollisionsfrei ist, auch wenn sich temporäre Hindernisse, z.B. Menschen oder andere AMR, in der Umgebung aufhalten.
- Minimale Gesamtlänge: Minimale Gesamtlänge aller Teilstücke für die Wegführung vom Start S bis zum Ziel G.
- Minimale Fahrtzeit: Der AMR soll das Ziel möglichst schnell erreichen können. Das bedeutet: er soll die Teilstücke möglichst schnell befahren können, die Anzahl der Teilstücke soll möglichst gering sein und die Übergänge der Teilstücke sollen sehr weich sein.
- geringe Gesamtrotation: Es sollen keine großen Richtungsänderungen notwendig sein.
- minimale Rechenzeit für die Wegbestimmung: wenn sich die Hindernisse andauernd ändern (z.B. wenn sich viele Menschen in einem Raum befinden), ist es wichtig, schnell Ausweichstrecken zu finden.
- Sicherheitsaspekte: Ist in einem Gebiet die Positionsbestimmung schwierig (z.B. eine Glasscheibe bei der Laserabtastung, schwere Vorhänge bei Ultraschall usw.), so kann es sein, dass der AMR die Orientierung verliert und in gefährliche Situationen (Treppe, Löcher, Sand usw.) kommt, die für den AMR und die in der Nähe befindlichen Menschen oder Maschinen gefährlich werden können.
- Zielgerichtes Arbeiten auch bei unbekannter Umgebung: Es soll auch bei zunächst unvollständig erfasster Umgebung ein Pfad gefunden werden.

Da der AMR für die Pfadsuche auf eine Umgebungsdarstellung angewiesen ist und der Schwierigkeitsgrad der Pfadsuche von der Anzahl der Hindernisse in einer Umgebung abhängig ist, wollen wir nachfolgend einige Darstellungformen betrachten.

2.1 Freiraumdarstellung durch Konfiguratiosraum Lozano-Perez

Bei der Freiraum-Darstellung wird der freie bzw. befahrbare Bereich modelliert. Das Komplement der Freiräume ergibt dann die Hindernisfläche bzw. die nicht befahrbaren Bereiche. Ein Vertreter der Freiraumdarstellung ist Lozano-Perez ([LoPe81], [LoPe83]) mit der Configuration Space Methode.

Dabei wird ein kreisrunder AMR verwendet und alle Hindernisse werden um die halbe AMR Breite $m_B/2$ aufgebläht. Damit kann der Roboter auf einen Punkt reduziert werden. Die übrig bleibenden Freiräume sind für den Punkt befahrbar.

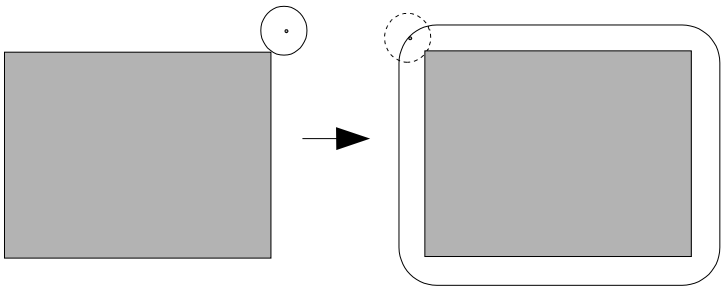


Abbildung 1: Lozano- Perez configuration space Methode

Für die Pfadsuche wird die freie Fläche in konvexe Polygone unterteilt und davon dann ein Suchbaum erstellt, wobei jedes Polygon einen Knoten und jedes umittelbare Nachbarpolygon eine Kante darstellt. Die Suche im Graphen erfolgt dann mit dem effizenten A* Algoritmus ([Hart68]).

Nachteile des Verfahrens:

- Ausgangspunkt ist eine bekannte Umgebung
- Hindernisse werden durch das Aufblähen komplexer.
- Der AMR wird auf einen Kreis idalisiert.

Eine Verfeinerung des Verfahrens stellt das Aufblähen der Hindernisse auf die AMR Breite (von Tsu-Chuan Chang [Tsu86]) dar. Dabei wird der AMR auf eine Strecke reduziert.

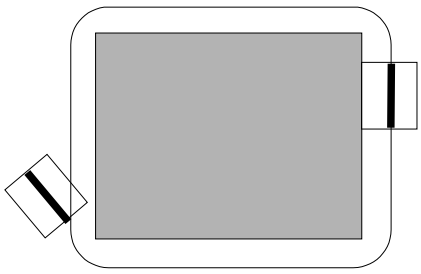


Abbildung 2: AMR Reduzierung auf eine Linie

Der Vorteil gegenüber Lozano-Perez ist hier, dass der AMR nicht mehr auf einen Kreis idealsiert werden muß und damit mehr Freiräume zur Verfügung stehen.

2.2 Freiraumdarstellung durch konvexe Regionen wie in [Crowley85].

Crowley beschreibt dabei die Freiräume durch konvexe Regionen so, dass die Kante zwischen zwei solchen Regionen die Knoten und die Strecken zwischen den Knoten innerhalb einer Region die Kanten darstellen. Um die Abmessungen des AMR zu berücksichtigen, werden die konvexen Regionen um die AMR-Breite geschrumpft und die dadurch enstehenden Lücken zwischen den zuvor benachbarten Polygonen durch sogenannte doorways (müssen nicht konvex sein) verbunden. Dieser Graph stellt dann die befahrbaren Strecken dar. Die Pfadsuche erfolgt mit dem Dijkstra Algorithmus([Aho74], [Aho83]).

Nachteil

- Die Karte muß vollständig vorhanden sein, da nur dann die konvexen Regionen erstellt werden können.
- Das Erstellen der konvexen Regionen erfolgt meist durch Voronoi Diagramme, deren Erstellung rechenintensiv ist.

2.3 Hindernisdarstellung durch Kreise (Moravec)

Bei der Hindernisdarstellung werden die Hindernisse bzw. die nicht befahrbaren Bereiche der Umgebung modelliert. Das Komplement dieser Hindernisbereiche ergibt dann die befahrbare Fläche.
Moravec verwendet für die Darstellung der Hindernisse Kreise, die die Hindernisse einschliessen. Dadurch wird das Aufblasen der Hindernisse, wie es Lozano- Perez verwendet, besonders einfach. Die Pfadsuche beschränkt sich nun darauf, innerhalb der Darstellung einen direkten Weg von Start S nach Ziel G zu fahren und auftretende Hindernisse mit einer von vier Möglichkeiten Abbildung 3 zu umfahren.

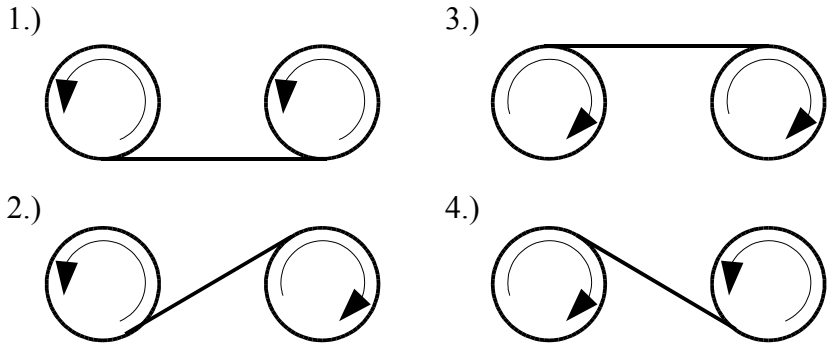


Abbildung 3: Die 4 möglichen Wegführungen bei Moravec

Die Pfadsuche erfolgt mit Hilfe eines Shortest Path Algorithms, wobei Hindernisse die Knoten und Verbindungslinien die Kanten darstellen.

Vorteile:

- einfache Modellierung
- einfache Pfadfindung

Nachteile:

- Mit der Modellierung der Hindernisse durch Kreise geht Freiraum verloren, sodass unter Umständen das Auffinden der Pfadführung unmöglich wird.
- Ungeeignet für eine strukurierte Umgebung mit vielen rechten Winkeln

2.4 Potentialfelder (Freyberger)

Beim Potentialfeldverfahren üben die Hindernisse abstoßende Kräfte und das Ziel anziehende Kräfte auf den AMR aus. Die abstoßende Kraft eines Hindernisses wird um so größer, je näher der AMR dem Hindernis kommt und geht gegen unendlich an der Hindernisgrenze. Dadurch wird eine Kolision mit dem AMR ausgeschlossen. Überlagert man nun die Potentialfelder so ergibt sich ein Gesamtfeld welches die Richtung für den AMR bestimmt.

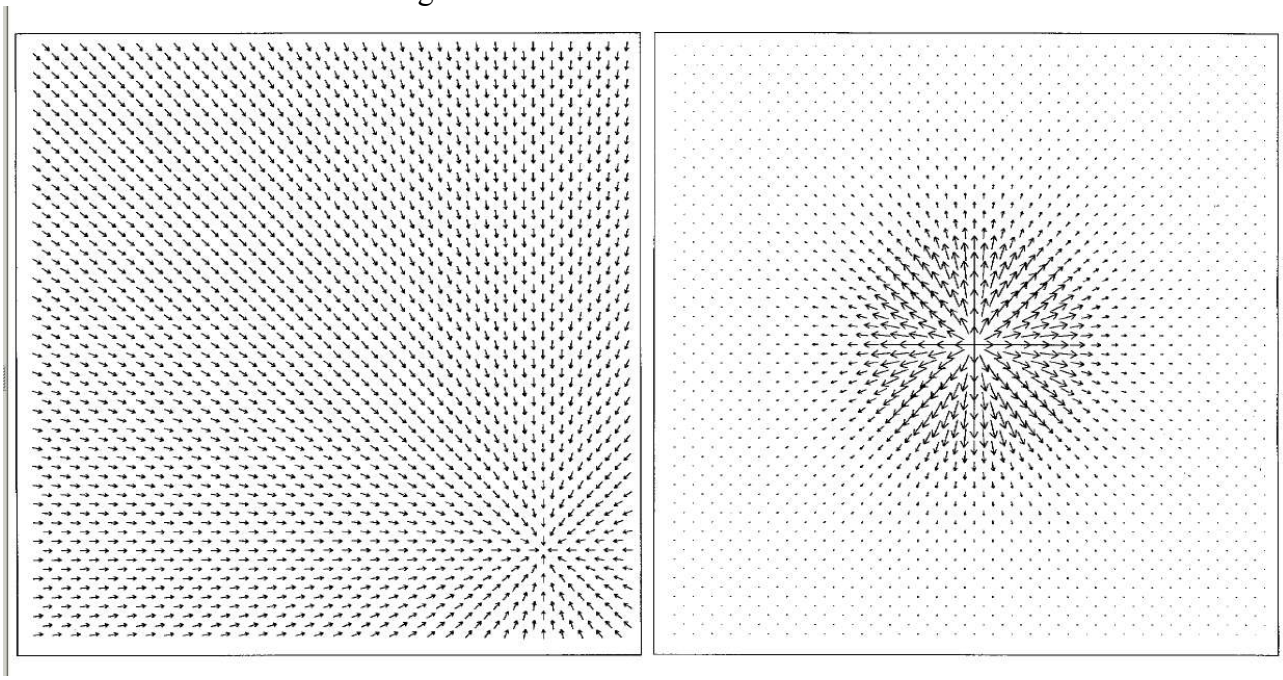


Abbildung 4: Anziehendes Ziel links, abstoßendes Hindernis rechts

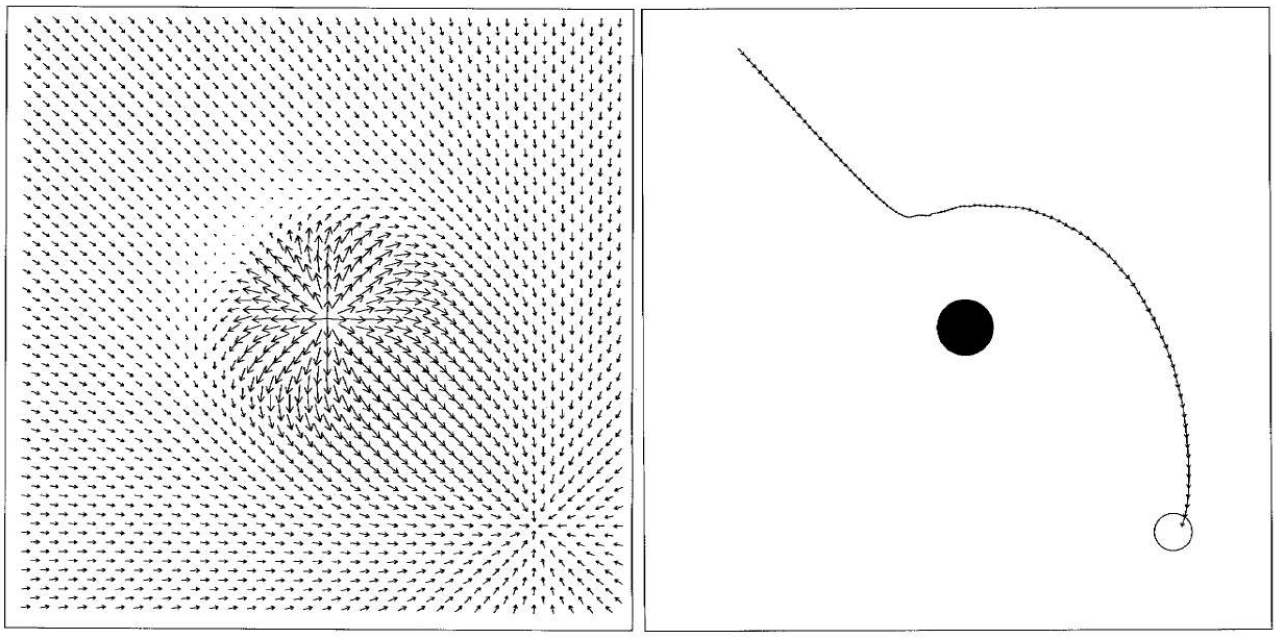


Abbildung 5: Gesamtpotentialfeld aus Überlagerung links, Pfadführung um das Hindernis rechts

Das Potentialfeldverfahren liefern damit bei relativ geringem Rechenaufwand auf Basis von geringen Sensordaten Steuerkurse die die kinematischen Gesetzmäßigkeiten des AMR berücksichtigen. Beim Potentialfeldverfahren ist immer die Potentialfunktion in Abhängigkeit der Hindernisse das bestimmende Kriterium. Verschiedene Ansätze für Potentialfelder unterscheiden sich deshalb in der Bestimmung dieser Funktion ([Arkin98], [Khatib86]).

Vorteile:

- Einfache Darstellung von Hindernissen
- Einfache Pfadfindung durch Folgen des Feldes

Nachteile:

- Ein entscheidener Nachteil ist der, dass es lokale Minima geben kann (Sackgassen), in den sich der AMR verfängt.
- Zyklisches Verhalten aufgrund von lokalen Minima

2.5 Rasterdarstellung der Umgebung

Bei der Rasterdarstellung wird die Umgebung in ein Rechteckraster mit einer vordefinierten Breite unterteilt, wobei jedes Rechteck das Attribut [„frei“, „nichtfrei“, „nicht getestet“] haben kann.

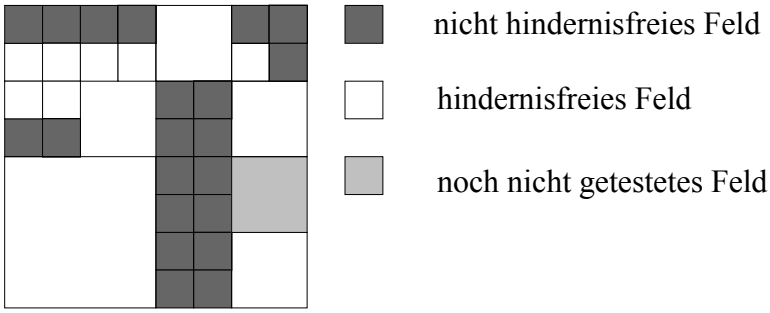


Abbildung 6: Rasterdarstellung einer Umgebung

Eine solche Darstellung wird häufig in Computerspielen eingesetzt, da dort die Genauigkeit eine kleinere Rolle spielt. Hindernisse sind an der Rasterdarstellung angepasst. In der realen Welt hingegen halten sich Hindernisse nicht an das Schachbrettmuster.

Nachteile:

- Die Bewegungsfreiheit beschränkt sich auf die 4 Richtungen: Nach oben, nach unten, nach rechts, nach links .
- Die Genauigkeit ist auf den Rasterabstand beschränkt.
- Leere Räume benötigen Speicher für die Rasterdarstellung und dieser steigt mit O(k*n) mit k Anzahl der Rasterschritte in y Richtung und k Anzahl der Schritte in x Richtung.
- Das Einfügen und Entfernen von Hindernissen ist sehr zeitaufwendig.

Den Speicherbedarf kann sich bei geringer Hindernisdichte durch eine spezielle Baumstruktur verringern. Dabei wird ein vierblättriger Baum mit nicht voll besetzten Ästen verwendet. Ist ein Ast nicht voll besetzt, heißt das, dass bei einem schwarz eingefärbten Knoten alle Äste Hindernisse enthalten, bei weißem Knoten keine Hindernisse in den Ästen vorhanden sind und bei grauer Einfärbung der Zweig sowohl Freiräume als auch Hindernisräume enhalten kann.

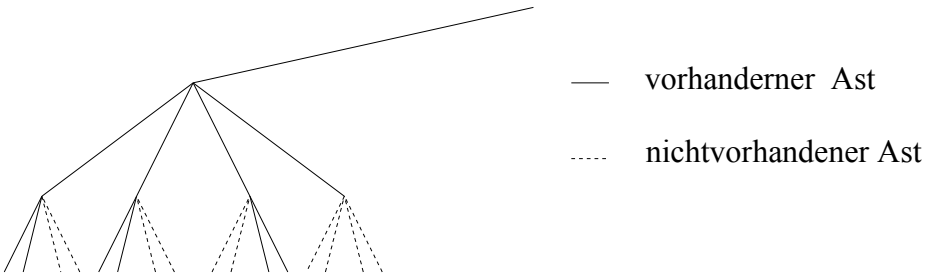


Abbildung 7: Baumstruktur bei einer Rasterdarstellung

Vorteile:

1. Die Suche eines Pfades beschänkt sich auf der Suche nach dem Pfad in einem Graphen z.B. mit dem A*Algoritmus.
2. der Aufbau der Simulationsumgebung ist relativ einfach.

Eines der Einsatzgebiete für Rasterdarstellung sind Computerspiele wo man die Umwelt nach dem Raster modeliert und nicht umgekehrt wie in der realen Welt.

2.6 Beschreibung einer Simulationsumgebung mit geometrischen Figuren

Hier wird die Umgebung MAP durch einfache geometrische Figuren, z.B. Linien, Kreise, Rechtecke,und Polygone, dargestellt. Meist beschränkt man sich dann aber auf Linen für die Hindernisdarstellung um die Berechnungen einfach zu halten. Das größte Problem dabei ist es, effizent festzustellen, wo sich Hindernisse befinden und wo nicht. Dafür werden meist spezielle Baumstrukturen als Speicherstrukturen verwendet. Die damit bessere Effizienz beim Überprüfen der Hindernisfreiheit bzw. beim Suchen nach Hindernissen innerhalb einer vorgegeben Fläche wird hier mit einem höheren Speicheraufwand erkauft.

Nachteile:

1. Der Test, ob eine Fläche bei gegebener MAP hindernisfrei ist oder nicht, ist aufweniger als bei einer Rasterdarstellung
2. Die Pfadsuche gestaltet sich komplizierter als bei der Rasterdarstellung
3. Das Einfügen und Löschen der Hindernisse ist aufwendig

Vorteile:

1. Die Pfadrichtung ist nicht durch die Darstellung der Umgebung eingeschränkt
2. Die Pfadführung kann genauer berechnet werden.

2.7 Visibility Graph sich nicht schneidender Polygone als Hindernisse

Beim Visibility Graphen handelt es ich um einen Graphen, bei dem Eckpunkte der Polygonhindernisse als Knoten dargestellt werden und alle Verbindungstrecken zwischen zwei beliebigen, verschiedenen Knoten, die keine Begrenzungslinie eines der Polgone schneiden, als Kanten dargestellt werden. Kanten zwischen den Eckpunkten einer Begrenzungslinie gehören dabei zum Graphen. Ein solcher Graph mit einem Startpunkt S und einem Zielpunkt G ist in Abbildung 8 zu sehen. Das Pfadsuchen in diesem Graphen erfolgt mit dem Dijkstra SP Algorithmus. Dieser Graph lässt sich laut [Kit03] mit dem Algorithmus von Ghosh & Mount mit E als Anzahl der Ecken in $O(n*\log n+E)$ berechnen.

Vorteile:

- der Algorithmus liefert den kürzesten Pfad

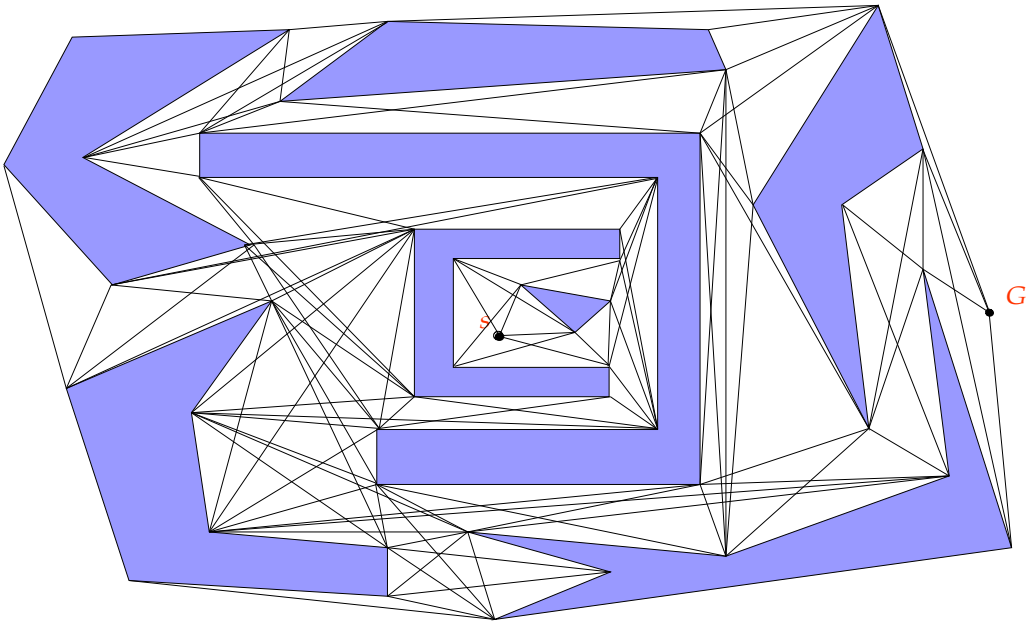


Abbildung 8: Visibility Graph

3 Theoretische Grundlagen zur geometrischen Pfadplanung

Die nachfolgende Umsetzung setzt auf einen ähnlichen Ansatz wie beim Sichbarkeitsgraphen, wobei die Knoten aber nicht die Endpunkte der Hindernisstecken sind, sondern spezielle hindernisfreie Ausweichpunkte, die auf der Geradenverlängerung der Hinderstecke liegen. Mit Hilfe des A* Algorithmus erhoffen wir uns in Standardsituationen eine schnellere Pfadfindung, da nicht alle möglichen Wegstrecken berechnet werden müssen. Da wir auf das Aufblähen der Hindernisse wie bei Lozano-Perez verzichten, erfolgt nach der Wegplannung die eingentliche Pfadplannung mit Hilfe von sogenannten Bubbles. Dabei werden entlang der Wegführung Bubbles (Kreise) so plaziert, dass sie einen Punkt der Wegstrecken enthalten und sich gegenseitig scheiden, sodass eine Folge dieser Bubbles eine Pfadführung ergibt. Einen Methode die mit Bubbles arbeitet ist die Pfadoptimierung mit Hilfe eines Gummibandes [QuKh93]. Ausgehend von einem gefunden Pfad wird dabei mitt virtuellen Kräften der Pfad so verformt, daß dieser den nöten Sichheitabstand zu den Hindernissen hat und gleichzeitig eine Glättung des Pfades entsteht. Möglichst große sich schneidende Bubbles die entlang des Gummibands plaziert werden und deren Mittelpunkt auf dem Gummiband liegt ergeben dann einen befahrbaren Pfad. Innerhalb der Bubble können dann bei auftrehten von Hindernissen während der Fahrt locale Ausweichstrecken gefunden werden.

3.1 Prinzipielle Idee bei der geometrischen Pfadsuche

Die geometrische Pfadplanung beruht auf einer einfache Idee:

„Wenn wir nicht durch das Hindernis hindurchfahren können, dann müssen wir um das Hindernis herumfahren.“

Die Idee soll anhand eines Beispiels erläutert werden . Dazu wird eine Umgebungsbeschreibung benutzt, die ausschließlich durch Hindernisse (Linien) beschrieben wird. Es werden alle Hindernisse in eine Karte (GM) eingetragen und die Startposition (S) und Zielposition (G) festgelegt.

Auf diese Karte wird nun ein Algorithmus angewandt der im wesentlichen aus 4 Phasen besteht:

Phase 1: Preprozessing

Phase 2: Aufbau des Wegnetzes

Phase 3: Ermittlung des kürzesten Weges

Phase 4: Feinplanung der Kurssegmente

In der Phase 1 werden zu jeder Hindernisstrecke der Karte Wegpunkte generiert, die später dazu benutzt werden, um diese Hindernisse herumzufahren. Dazu wird in der Verlängerungen jeder Hindernisstrecke an beiden Enden ein Wegpunkt angelegt. Die Wegpunkte werden so konstruiert, dass im Umkreis mit Radius m_R dieser Punkte keine Hindernisse liegen. Wenn die Breite des Roboters in Fahrtrichtung mit m_B angegeben ist, so muss m_R größer oder gleich $m_B/2$ sein.

In Phase 2 wird ein Wegenetz aufgebaut. Dabei handelt es sich um einen Graphen mit den Wegpunkten als Knoten und den Wegstrecken als Kanten. Die Kanten in diesen Graphen stellen Wegstrecken dar, die auf den ersten Blick befahrbar erscheinen. Zu diesen Zeitpunkt wird nur sichergestellt, dass die Wegstrecke kein Hindernis schneidet. Es ist also nicht abgetestet, ob links und rechts der Wegstrecke auch genügend Platz vorhanden ist, um den Weg mit dem Roboter abfahren zu können. Dieser Test wird erst in Phase 4 durchgeführt, da sich dort die Anzahl der Wegstrecken auf die eigentliche Wegführung zum Ziel reduziert. Damit lässt sich der Aufwand für den Test erheblich reduzieren.
Diese Art der Wegplanung entspricht auch der Vorgangsweise, nach der ein Mensch einen Weg planen würde. Dieser wird auch zuerst die Karte betrachten, um sich einen grundsätzlich eine Idee für die Wegführung zu machen, diese dann zu verifizieren und gegebenfalls abzuändern.

Der Grundgedanke zum Aufbau des Wegnetzes ist der, ein gegebenes Kurssegment daraufhin zu untersuchen, ob dieses ein Hindernis schneidet oder nicht. Dazu wird zu Beginn das Wegenetz mit der Wegstrecke, die S und G verbindet initialisiert. Diese Wegstrecke erhält das Attribut „zu testen“. Wird ein Schnittpunkt einer zu testenden Wegstrecke mit einem Hindernis festgestellt, so werden die Wegpunkte, die in Phase 1 zu diesem Hindernis erzeugt wurden, herangezogen, um Ausweichstrecken anzulegen, sofern diese noch nicht vorhanden sind. Diese Ausweichstrecken verbinden jeweils den Anfangspunkt und Endpunkt des zu testenden Weges mit den beiden Wegpunkten des geschnittenen Hindernisses. Auch diese Wegstrecken erhalten das Attribut „zu testen“. Es werden nun solange Wegstrecken mit den Attribut „zu testen“ überprüft, bis entweder eine Verbindung vom Start zum Ziel gefunden oder keine Wegstrecke mehr zu testen ist. Schneidet eine zu testende Wegstrecke kein Hindernis, zu wird sie als „befahrbar“ markiert.
Liegt das Wegenetz vor, so wird in Phase 3 der günstigste Weg von S nach G durch das Wegenetz ermittelt.

In der Phase 4 wird die Wegführung daraufhin untersucht, ob die Teilstrecken für den Roboter unter Berücksichtigung der Abmessungen des Roboters befahrbar sind. Dazu wird eine Rechteck mit der Breite m_B des Roboter über die Teilstrecke gelegt und getestet ob dieses hindernisfrei ist. Wird ein Hindernis gefunden das den Weg scheidet, so wird versucht, dieses Hindernis durch Hinzufügen eines zusätzlichen Wegpunktes zu umfahren und dann auf den ursprünglichen Weg zurückzukehren. Ist auch dies nicht möglich, weil eine Wegstrecke durch eine zu enge Stelle führt, wird die Wegstrecke im Wegenetz als nicht befahrbar markiert und bei Phase 3 fortgesetzt.

4 Die geometrische Pfadplanung im Detail

Hier sollen nun die oben beschriebenen Phasen im Detail betrachtet werden. Um eine klare Definition zu erhalten, wird zunächst eine Notation definiert.

4.1 Notation

Grundlage aller Betrachtungen ist ein zweidimensionales Koordinatensystem, in dem alle Positionen der Wegpunkte, Hindernisse und des AMR (Autonomer Mobiler Roboter) festgelegt sind

Die Position p ist definiert als das Tupel

$$p=(x,y)\quad x,y\in\mathbb{R}$$

Die Menge P aller Positionen p ist definiert als

$$P=\{p\,|\,p\,ist\,Position\},\ P\subset\mathbb{R}\,x\,\mathbb{R}$$

Mit Menge Q=P x P werden alle möglichen Verbindungsstrecken zwischen den Positionen aus P definiert.

Das Prädikat

$$s:Q\,x\,Q\rightarrow\{true,\,false\}$$

sei definiert als

$$\forall\, q_1,q_2\in Q: s(q_1,q_2)::=q_1\,schneidet\,q_2$$

und gibt damit an, ob sich die beiden Strecken q_1 und q_2 schneiden.

q_1 und q_2 lassen sich durch die Geradengleichungen q_1: $y=k_1*x+d_1$ und q_2: $y=k_2*x+d_2$ darstellen?

Für den Schnittpunkt (x_s,y_s) gilt dann

$$k_1\cdot x_s+d_1=k_2\cdot x_s+d_2$$
$$x_s=\frac{d2-d1}{k1-k2}$$
$$y_s=k_1\cdot x_s+d_1$$

Die Länge einer Strecke $q\in Q$ sei durch die Funktion

$$l:\ Q\rightarrow\mathbb{R}$$

definiert.

Seien $p_1,p_2\in P$ *mit* $p1=(x_1,y_1)$, $p2=(x_2,y_2)$ *und* $q=(q_1,q_2)\in Q$ dann ist

$$l(q)::\sqrt{(x_1-x_2)^2+(y_1+y_2)^2}\ .$$

Die Hindernisstrecken o sind definiert als das Tupel von Positionswerten $(p_1,p_2)\in P$.

Die geometrische Karte MAP ist definiert als

$$MAP=\{o\ \mid\ o\ \ ist\,eine\,Hindernisstrecke\}$$

Die Planungsaufgabe wird jetzt durch die Vorgaben des Startpunktes S und des Zielpunkts G mit $S, G \in P$ beschrieben.

Das durch die Wegplanung aufzubauende Wegenetz WN wird definiert durch ein n-Tupel

$WN=(N,E,f_N,f_E,f_S,f_G)$,

das einen Graphen darstellt.

N ist die Menge der Knoten des Graphen, die als Wegpunkte bezeichnet werden.

$N \subset P$.

E ist die Menge der Kanten des Graphen, die als Wegstrecken bezeichnet werden.

$E \subseteq N \times N$

Zusätzlich werden Funktionen definiert, die wichtige Informationen über die Elemente von N bzw. E liefern.

Die Funktion

$f_E : E \rightarrow \{T, D, X\}$

ordnet jedem $e \in E$ ein Attribut zu, das Auskunft über die Befahrbarkeit der Wegstrecke gibt.

$f_E(e) ::= T$: e ist noch auf Schnitte mit $o \in MAP$ zu testen
D: e wurde getestet und ist befahrbar, d.h. $\neg \exists o \in MAP.s(o, e)$
X: e wurde getestet und ist nicht befahrbar, d.h. $\neg \exists o \in MAP.s(o, e)$

Wege $e \in E$ mit $f_E(e)=T$ werden im Folgenden als Wegstreckenkandidaten bezeichnet, da für sie noch abzutesten ist ab sie befahrbar sind oder nicht.

Allgemein definiert wird die Funktion

$f_D : N \times N \rightarrow \mathbb{R}$

die die Kosten angibt, um vom Wegpunkt $x_1 \in N$ zu einem anderen Wegpunkt $x_2 \in N$ zu gelangen, als

$$f_D(x_1, x_2) ::= \begin{cases} 0, & falls\ x_1 = x_2 \\ & falls\ \exists (e_1, e_2, \ldots, e_n)\ mit\ e_i = (a_i, b_i) \in E \\ < \infty, & f_E(e_i) = D,\ a_1 = x_1,\ b_n = x_2\ und\ a_i = b_{i-1},\ d.h.\ es\ gibt \\ & eine\ Folge\ von\ befahrbaren\ Strecken\ die\ von\ x_1\ nach\ x_2\ führen, \\ \infty, & sonst \end{cases}$$

Der konkrete Wert von f_D wird über eine Heuristik gebildet. Je nachdem, welches Optimalitätskriterium die gefundene Wegführung erfüllen soll.

Die Funktion

$f_S : N \rightarrow \mathbb{R}$

gibt die Kosten an, um von Start S zum einem Wegpunkt x zu gelangen und ist definiert als

$$f_s(x) ::= \begin{cases} 0, & falls\ x = S \end{cases}$$

$< \infty,$ $\quad falls \exists (e_1, e_2, \ldots, e_n) mit\ e_i = (a_i, b_i) \in E,$
$f_E(e_i) = D, a_1 = S, b_n = x\ und\ a_i = b_{i-1}$

$\infty,$ $\quad sonst$

Damit ist $f_S(x)=f_D(S,x)$.

Analog gibt die Funktion

$$f_G: N \to \mathbb{R}$$

die Kosten an, um vom Wegpunkt x aus zum Ziel G zu gelangen. Sie ist definiert als

$f_G(x) ::= \quad 0, \quad falls\ x = G$
$< \infty,$ $\quad falls \exists (e_1, e_2, \ldots, e_n) mit\ e_i = (a_i, b_i) \in E,$
$f_E(e_i) = D, a_1 = x, b_n = G\ und\ a_i = b_{i-1}$

$\infty,$ $\quad sonst$

also ist $f_G(x)=f_D(x,G)$.

Die Funktion

$$f_N: N \times N \to \{true, false\}$$

unterteilt die Menge N der Wegpunkte in Klassen untereinander erreichbarer Punkte. Für zwei Knoten gilt $f_N(x_1, x_2) = true$, wenn es eine Folge von Strecken $e_1, e_2, \ldots, e_n \in E$ gibt, die befahrbar sind und über die x_2 von x_1 aus erreicht werden kann. Aufbauend auf der Definition von f_D kann also f_N definiert werden als

$$f_N(x_1, x_2) ::= (f_D(x_1, x_2) \neq \infty) \quad .$$

Ergebnis der Berechnungen der Wegplanung ist der Kurs W, der eine Folge von Wegstrecken darstellt, die hindernisfrei sind und von S nach G führen.

$$W = (w_1, w_2, \ldots, w_n), \quad w_i \in E, \quad f_E(w_i) = D$$

W ist die leere Folge (), wenn keine solche Wegführung von S nach G existiert.

Die Pfadplanung hat gewisse Merkmale des AMR zu berücksichtigen. Dabei werden folgende physikalische Parameter berücksichtigt:

r_B	Breite des AMR in Fahrtrichtung
r_L	Länge des AMR
r_{min}	der Minimale Wenderadius
s_A	angestrebter Sicherheitsabstand

Für den Pfad wird ausgehend von den obigen Parametern folgende Pfadbeschaffenheit angestrebt.

- Einhaltung einer minimale Korridorbreite m_B
- Einhaltung des Kurvenradius zwischen den Pfadsegmenten

Auch wenn diese Forderungen auf den ersten Blick nicht sehr komplex aussehen, so gehört das Problem „curvature constraint path planning“ zu den NP-hard Problemstellungen.

4.2 Wegplanungsalgorithmus

Auf der oben getroffenen Festlegungen kann nun der Wegplanungsalgorithmus formuliert werden:

Wegplanung:

```
input S,G,MAP;
output W;
var WN;
   begin

   konstruiere Wegpunkte;                                    //Preprozesseing
   initialisiere Wegenetz;

   repeat

      Wegenetzkonstruktion:
      while not fertig do                                    //Abbruchstrategie
      begin
         Wähle_Wegstrecken_Kandidat                          //Präferenzstrategie
         if not befahrbar(x) then                            //Befahrbarkeitstest
            ermittele_Ausweichstrecken(x)                    //Ausweichstrategie
         else
            f_E(x)::=D;
         endif;
      end;

      Wegbestimmung:
      Vervollständige_Wegenetz;                              //Vervollständigung
      Ermittle_kürzesten_Kurs_zum _Ziel;                     //Suchstrategie
   until verifiziert                                         //Feinplanung &Verifikation

end Wegplanung;
```

Dieser Algorithmus implementiert das oben beschriebene Verfahren. Die einzelnen Zeilen beschreiben dabei jeweils ein grundlegendes Konzept. Dieses soll nun der Reihe nach im Detail betrachtet werden.

4.3 Preprozessing

Aufgabe des Preprozessing ist, die über die Umwelt bekannten Informationen (Hindernisse) in die MAP einzutragen. Zusätzlich werden zu jedem Hindernis die beiden dazugehörigen Ausweichpunkte konstruiert, welche später zur Konstruktion des Wegenetzes verwendet werden. Zudem wird eine Liste der zu „testenden“ Wegstrecken des Wegenetzes mit der Wegstrecke (S,G) initialisiert.

4.4 Konstruktion des Wegenetzes - Grobplanung

Nachdem durch das Preprozessing die Datenstruktur MAP erzeugt wurde, ist es die Aufgabe der Wegenetzkonstruktion ein Wegenetz aufzubauen das mindestens einen Weg W vom Start S zum Ziel G enthält. Wichtigste Aufgabe ist es eine „grobe“ Idee eines Weges von S noch G zu erhalten um diese dann in der Folge bei der Feinplanung zu verifizieren zu können. Dabei wird nur überprüft

ob die Wegführung keine unpassierbaren Hindernisse schneidet. Der Zweck einer solchen Vorgangsweise besteht darin, die rechenintensive Überprüfung der Hindernisfreiheit eines Korridors auf die eigentliche Wegführung zu reduzieren. Deshalb wird die Konstruktion des Weges W im folgenden auch als Grobplanung bezeichnet. Wird ein Weg durch die Feinplanung verworfen, so wird die Grobplanung erneut angestoßen bis entweder ein Weg gefunden wird, der einer Verifizierung durch die Feinplanung standhält oder die Grobplanung fehlschlägt (repeat ..until -Schleife).

Bevor wir aber zur Feinplanung übergehen wollen wir uns den Teil der Grobplanung noch etwas genauer anschauen:

```
Wegenetzkonstruktion:
while not fertig do                                    //Abbruchstrategie
begin
   Wähle_Wegstrecken_Kandidat                          //Präferenzstrategie
   if not befahrbar(x) then                            //Befahrbarkeitstest
      ermittele_Ausweichstrecken(x)                    //Ausweichstrategie
   else
      f_E(x)::=D;
   endif;
end;
```

4.5 Abbruchstrategie

Das Prädikat „fertig“ steuert den Abbruch bei der Wegenetzkonstruktion innerhalb der while Schleife. Für die Implementation dieses Prädikates sind verschiede Ansätze möglich.

4.5.1 Spätest möglicher Abbruch

Es bestehen keine Möglichkeiten mehr, zusätzliche Wege zu finden da es keine Wegstrecken-Kandidaten mehr gibt, die „zu testen“ wären. Eine solches Abbruchkriterium könnte so aussehen.

function fertig_a: boolean;

begin
fertig_a:= $\neg\exists e\in E.\, f_E(e)=T$

end;

Unter diesem Abbruchkriterium wird der Algorithmus sicher terminieren da die Anzahl der möglichen Wegstrecken durch die begrenzte Anzahl der Wegpunkte auch begrenzt sein muss.

4.5.2Frühest möglicher Abbruch

Frühest möglicher Abbruch bedeutet: es wurde eine Wegführung zwischen S und G gefunden, die keine Hindernisse schneidet. Das Kriterium lässt sich wie folgt formalisieren.

function fertig_b:boolean;

begin
fertig_b:= $f_S(g)<\infty$ or fertig_a;
end;

Um einen Abbruch sicherzustellen muss fertig_a hinzugefügt werden, da sonst der Algorithmus im Falle, dass kein Weg existiert nicht abbrechen würde.

4.5.3 intelligenter Abbruch

Es wurden ein oder mehrere Wegführungen gefunden und der Aufwand zur Ermittelung weiterer Lösungen erscheint, bezogen auf die Qualität einer solchen Lösung, nicht mehr gerechtfertigt.

Dieses Kriterium lässt sich aber ohne Kenntnis einer Präferenzstrategie nicht formalisieren. Für eine Präferenzstrategie ist nötig, für eine bereits gefundene Lösung, eine konkrete Güte festzulegen. Erst dann ist es möglich zwei verschieden Lösungen gegeneinander abzuwiegen und eine Entscheidung für einen Abbruch zu fällen.

4.5.4 Präferenzstrategie

Für einen Intelligenten (zielgerichteten) Aufbau eines Wegenetzes ist es von entscheidender Bedeutung in welcher Reihenfolge die Auswahl der Wegstreckenkandidaten $e \in E \, mit \, f_E(e) = T$ erfolgt. Es ist bei diesem Verfahren zwar unerheblich in welcher Reihenfolge die Wegstreckenkandidaten getestet werden, solange der spät möglichste Abbruch gewählt wird. Ist jedoch die Umgebungsbeschreibung sehr komplex, so wird auch das Wegenetz entsprechend groß und enthält viele Wege die für die Wegführung irrelevant sind. Für einen möglichst frühen Abbruch sollte es aber Ziel sein, dass der gefundene Weg bereits einem gestellten Optimalitätskriterium gerecht wird. Zu diesem Zweck soll ein Verfahren vorgestellt werden das es ermöglicht günstige Wegstreckenkandidaten auszuwählen.

Für dieses Verfahren ist notwendig eine Kostenfunktion einzuführen die jedem $x \in E \, mit \, f_E(x) = T$ einen Wert zuordnet, der die Kosten für eine Fahrt von S über x nach G unter der Annahme abschätzt, dass x befahrbar ist.

Eine Präferenzstrategie wählt dann jeweils den Wegstreckenkandidaten aus, der die billigste Fahrt verspricht.

Es wird die Kostenfunktion

$$f_C : E \to \mathbb{R}$$

eingeführt.
Sei $x = (x_A, x_E) \in E$, sei $f_S(x_A) \leqslant f_S(x_E) \, bzw. \, f_G(x_E) \leqslant f_G(x_E), \, falls \, f_S(x_E) = f_S(x_E) = \infty$.

$f_C(x_E)$ liefert eine Abschätzung der Kosten einer Wegführung von S nach G über x nach der folgenden Vorschrift:

$$f_C(x) ::= f_S^*(x_A) + f_L(x_A, x_E) + f_G^*(x_E)$$

Grundidee ist, dass die Kostenfunktion aus drei Komponenten besteht, nämlich

1. die Kosten vom Start bis zum Anfangspunkt x_A von x
2. die Kosten von Anfangspunkt x_A nach x
3. die Kosten vom Endpunkt x_E von x zum Ziel G.

Die Kosten von x_A nach x_E sind je nach gewähltem Optimalitätskriterium sehr konkret zu bestimmen. Ist es z.B. Ziel, die kürzeste Wegführung zu finden, so ist $f_l(x_A, x_E) ::= l(x_A, x_E)$ zu definieren. Sollen die Anzahl der Wegstrecken bzw. Rotationen des AMR minimiert werden, so lässt sich dies in der Bildung von f_L über eine Konstante c berücksichtigen, etwa $f_L(x_A, x_E) ::= l(x_A, x_E) + c.$ Je größer der Wert c gewählt wird, desto mehr wird die Wegführungen zu tendenziell weniger Teilstrecken führen. Es gäbe noch eine Reihe weiter Möglichkeiten, weitere Qualitätskriterien zu definieren, die hier aber nicht besprochen werden .

Die Funktionen $f_S^* : N \to \mathbb{R} \, und \, f_G^* : \to \mathbb{R}$ stellen Abschätzungen für die Kosten vom Start S bis

zu einem Wegpunkt bzw. von dort bis zum Ziel G dar. Dabei wird auf die bereits definierten Funktionen f_S und f_G zurückgegriffen. Sei $x \in N$, dann ist

$$f_S^*(x) ::= \begin{array}{l} f_S(x),\ falls\ f_S(x) < \infty \\ c1 \cdot f_L(S,x) + c_2,\ \ sonst \end{array}$$

und analog

$$f_G^*(x) ::= \begin{array}{l} f_G(x),\ falls\ f_G(x) < \infty \\ c_1 \cdot f_L(x,G) + c_2,\ \ sonst \end{array}$$

Die beiden Funktionen liefern also für den Fall, dass x über befahrbare Wegstrecken mit S bzw. G verbunden ist, die dann bekannten tatsächlichen Kosten $f_S(x)$ bzw. $f_G(x)$ und andernfalls eine Abschätzung, die sich aus der Länge der Luftlinienverbindung zwischen S bzw. G und x ableiten. Die Tatsache, dass die Luftlinien im Normalfall kürzer sind als der endgültige Weg, wird durch die Konstante c_1 berücksichtigt. Durch ein entsprechend großes c_2 wird die Tatsache berücksichtigt, dass $f_S(x)$ bzw. $f_G(x)$ unendlich sind. Damit werden Strecken, die eine Verbindung zu S bzw. G besitzen, deutlich besser bewertet, was zu einer besseren Auswahl der Wegstreckenkandidaten führt.

Zusammenfassend noch einmal die Definition von f_C

$f_C(x)$	$f_G(x_e) < \infty$	$f_G(x_e) = \infty$
$f_S(x_a) < \infty$	$f_S(x_A) + f_L(x_A, x_E) + f_G(x_E)$	$f_S(x_A) + f_L(x_A, x_E) + c_1 \cdot f_L(x_E, G) + c_2$
$f_S(x_a) = \infty$	$c_1 \cdot f_L(S, x_A) + c_2 + f_L(x_A, x_E) + f_G(x_E)$	$c_1 \cdot f_L(S, x_A) + c_2 + f_L(x_A, x_E) + c_1 \cdot f_L(x_E, G) + c_2$

Nach der getroffen Festlegungen kann nun die Definition von f_D vervollständigt werden. Hier soll noch definiert werden wie f_D berechnet wird.

$$f_D(x_1, x_2) ::= \begin{array}{l} 0, \qquad falls\ x_1 = x_2 \\ \sum_{i=1}^{n} f_L(a_i, b_i),\ falss\ existes\ (e_1, e_2, \ldots, e_n)\ mit \\ \quad e_i = (a_i, b_i) \in E,\ f_E(e_i) = D,\ a_1 = x_1,\ b_n = x_2 \\ \quad und\ a_i = b_{i-1}\ und \sum_{i=1}^{n} f_L(a_i, b_i)\ \ minimal \\ \infty \ , \quad sonst \end{array}$$

Auf dieser Basis lässt sich die Präferenzstrategie algorithmisch wie folgt formalisieren :

procdure wähle_Wegstreckenkandidaten (var $x \in E$);

begin

$$bestimme\ x \in E\ mit\ f_E(x) = T\ und\ f_C(x)\ minimal\ d.h.$$
$$\forall e \in E.\ \ f_E(e) = T, e \neq x.\ \ f_C(e) \geq f_C(x)$$

end;

4.5.5 Befahrbarkeitstest

Aufgabe des Befahrbarkeitstests, der sich im Wegplanungsalgorithmus im Funktionsaufruf befahrbar widerspiegelt, ist es, einen Wegstreckenkandidaten die auf Schnitte mit Hindernisstrecken zu untersuchen. Wie schon angeführt, wird hier die Roboterbreite m_B nicht berücksichtigt, es geht hier nur um die Ermittlung einer groben Idee für die Wegführung von S nach G. Die detaillierte Behandlung unter Berücksichtigung der Breite findet erst bei der Verifikation und Feinplanung statt.

Ein Algorithmus für die Befahrbarkeit kann wie folgt aussehen:

function befahrbar($x \in E$):boolean;

begin
befahrbar:=true;
for all $o \in MAP$ do

if s(x,o) then befahrbar :=false;
endfor;

end;

Durch die Zentrale Stellung des Befahrbarkeitstests innerhalb des Wegplanungsalgorithmus ist leicht ersichtlich, dass dieser unmittelbar Einfluss auf das Laufzeitverhaltens hat. Deshalb wird hier bei Implementierung der MAP eine spezielle Datenstruktur verwendet, die Anzahl der Schittpunkttests wesentlich reduziert. Diese soll später betrachtet werden.

4.5.6 Ausweichstrategie

Wurde durch den Befahrbarkeitstest ein Wegstreckenkandidat als nicht befahrbar markiert, so ist es Aufgabe der folgenden Komponente des Wegplanungsalgorithmus, neue Wegstreckenkandidaten zu bestimmen um das betreffende Hindernis zu umfahren. Zur Erstellung dieser Ausweichstrecken werden die beiden Ausweichpunkte der Hindernisstrecke und der Start- und Zielpunkt des getesteten Wegstreckenkandidaten herangezogen.

4.5.6.1 Positionierung der Ausweichpunkte eines Hindernisses

Unter Berücksichtigung der AMR Abmessungen sollen die Ausweichpunkte so platziert sein, dass sie bei einem Schnitt einer Wegstrecke mit einer Hindernisstrecke und der daraus resultierenden Ausweichstrecken das geschnittene Hindernis nicht wieder schneiden. Dies lässt sich erreichen, indem man die Ausweichpunkte so konstruiert, dass diese an beiden Enden auf der Verlängerung des geschnittenen Hindernisses liegen. Weiteres müssen die Ausweichpunkte im Umkreis m_R hindernisfrei sein, was eine notwendige Bedingung darstellt, damit ein Pfad durch diesen Punkt vom AMR befahrbar ist.

Zum besseren Verständnis soll die Vorgangsweise anhand eines Beispiels veranschaulicht werden:

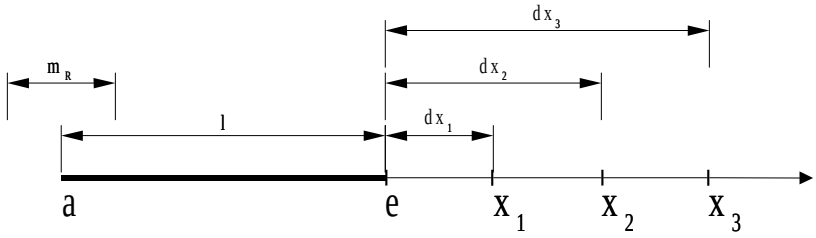


Abbildung 9: Erstellung eines Ausweichpunktes

Dazu betrachten wir ein Hindernis mit der Länge l =(a,e) und eine Roboterbreite m_B. Nun wird auf der Verlängerung des Hindernisses im Abstand $dx_1=m_R=m_B/2$ der Hilfspunkt x_1 konstruiert. Die Position x_1 wird nun daraufhin untersucht, ob sich im Umkreis mit Radius m_R kein Hindernis befindet. Existiert ein Hindernis o_1 innerhalb des Radius m_R so wird ein Punkt x_2 im Abstand dx_2 zu e so konstruiert, dass ein Kreis mit Radius m_R und Mittelpunkt x_2 das Hindernis o_1 gerade nicht mehr enthält. Dieser Vorgang wird solange wiederholt, bis ein Punkt gefunden wird, der einer Überprüfung standhält. Dieser Punkt ist dann der gesuchte Wegpunkt n_i.

Beispiel für die Konstuktion des Ausweichpunktes n_2

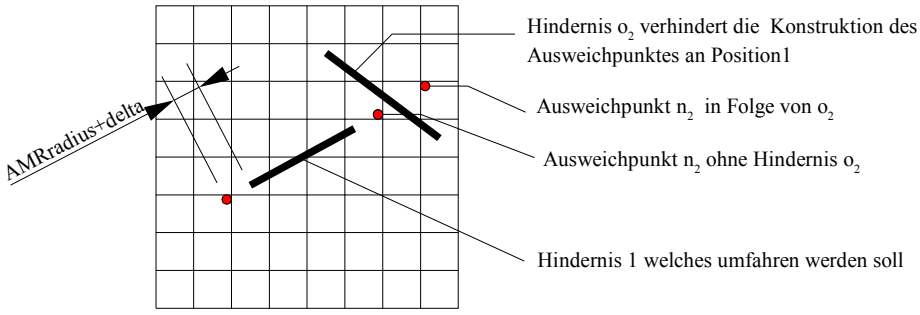


Abbildung 10: Beispiel für die Konstuktion des Ausweichpunktes n_2

4.6 Wegbestimmung

In Kapitel 4.4 wurde beschrieben wie das Wegenetz aufgebaut wird. Nun soll im Wegenetz die kürzeste Wegführung W_{min} im Graphen bestimmt werden, der S mit G verbindet. Dazu wird jeder Wegpunkt $n \in N$ mit zwei zusätzlichen Attributen (toStart,toGoal) versehen.

Daraus ergibt sich eine erweiterte Definition des Wegpunktes

n_E:=(n,toStart,toGoal) mit $n \in N ; toStart , toGoal \in \mathbb{R}$

und damit verbunden die Mengen E_E und N_E

Damit lässt sich nun die Suche des kürzesten Wegführung algorithmisch formulieren:

Ermittle_kürzeste Wegführung_zum Ziel

in N_E,E_E, f_E, S, G, f_S^*,f_G^*;

var i:integer;

var min:real;

var $N_1 \subseteq N_E$

var $n_{min} \in N_E$

$\forall n_E \in N_E \;\; toGoal := INF ; toStart := INF ;$

S.toStart:=0;

N_1:=N_E;

n_{min}:= $n \in N_E$ *mit n.toStart minimal*

while (n_{min} <> nil oder n_{min}=G)

$\forall e \in E_E$ *mit* $e = (n_{min} , x)$ x.toStart:=n_{min}.toStart+$f_C(n_{min},x)$

$N_1 := N_1$ *ohne* n_{min}

$n_{min} := n \in N_E$ *mit n.toStart minimal*

whileend;

if n <> G then W:=()

else

i:=0;w_i:=G;n_{min}:=G;

repeat

w_n:=n_{min}

bestimme $e \in E_E$ *mit* $e = (x , n_{min})$ *mit* $x.toStart = n_{min} .toStart$

n_{min}=e.x;

i:=i+1;

until n=S;

reverse W(w_i,.. ,w_0)

endif;

4.7 Vervollständigung des Wegnetzes

Nach der Wegenetzkonstruktion in 4.4 ist zwar eine Wegführung von S nach G möglich, diese ist jedoch unter Umständen mit Umwegen verbunden, wie das in der nachfolgenden Abbildung 11 dargestellt ist.

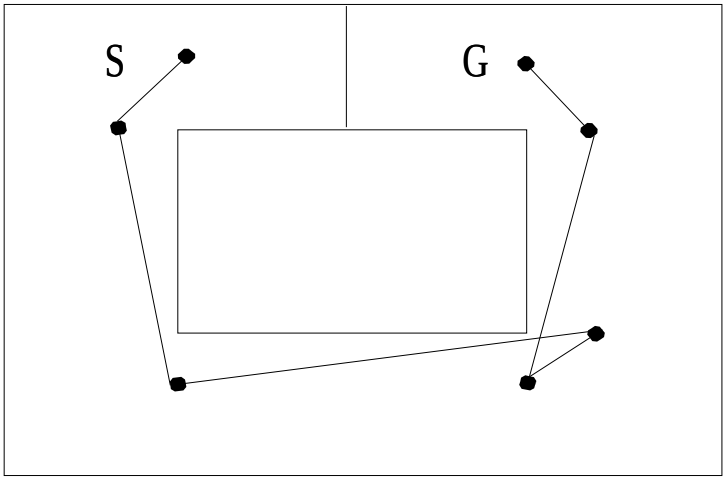


Abbildung 11: Wegführung mit Umwegen

Solche Umwege können entstehen wenn bei der Wegenetzkonstruktion als Abbruchstrategie der frühest mögliche Abbruch oder ein Intelligenter Abbruch gewählt wurde. Aus diesem Grund wird versucht das Wegenetz so zu ergänzen, dass diese Umwege eliminiert werden. Dazu werden jeweils zwei aufeinander folgende Wegstrecken $w_i=(n_{i-1},n_i),w_{i+1}=(n_i,n_{i+1})$ betrachtet und versucht, die Wegstrecken durch die alternative Wegstrecke $w_a:=(w_{i-1},w_{i+1})$ zu ersetzen. Dies ist immer dann möglich wenn e_a befahrbar ist. Damit ergibt sich für die Wegführung in Abbildung 11 eine neue Wegführung wie in Abbildung 12:

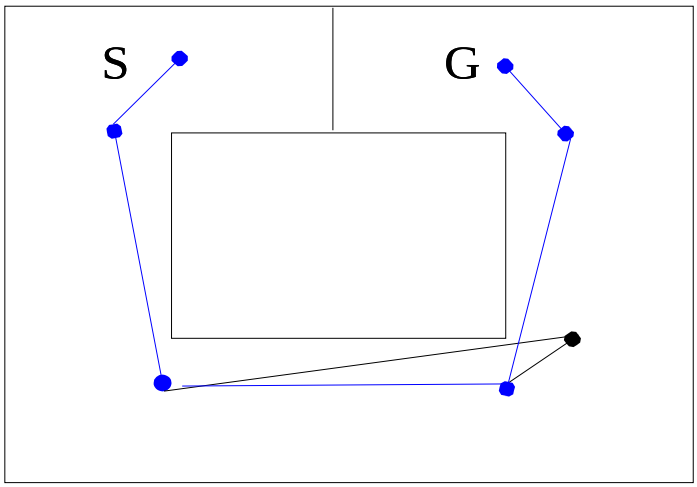


Abbildung 12: korregierte Wegführung

4.8 Verifikation und Feinplanung

In den vorherigen Kapiteln wurde beschrieben, wie das Wegenetz aufgebaut wird und wie innerhalb dieses Wegenetzes ein Kurs gesucht wird. Bei der Konstruktion des Wegenetzes wurde darauf geachtet, dass die Wegpunkte innerhalb eines Umkreises m_R Hindernis frei sind. Die Konstruktion der Wegstrecken erfolgte allerdings ohne die Berücksichtigung der AMR Abmessungen. Es ist Aufgabe der folgenden Komponente, die einzelnen Wegstrecken des Kurses zu verifizieren d.h. die tatsächliche Befahrbarkeit der Strecken zu überprüfen. Dazu wird diese Aufgabe in drei Abschnitte unterteilt:

- Engstellentest
- Korrektur der Wegführung wenn Hindernisse in die Wegführung hineinragen.

4.8.1 Engstellentest

Im folgenden wird eine Strecke q, die den minimalen Abstand zwischen zwei Hindernisstrecken markiert, und deren Länge kleiner als m_B ist, als Engstelle q_S bezeichnet. Scheidet eine Wegstrecke eine solche Engstelle so ist die Wegstrecke sicher nicht befahrbar. Um nun eine Engstelle in der Wegführung aufspüren zu können, wird über jede Wegstrecke der Wegführung ein Rechteck R mit der Breite $2*m_B$ wie im Abbildung 13 gelegt.

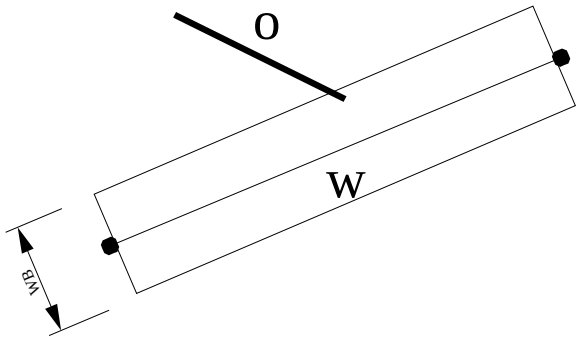


Abbildung 13: Engstellentest

Für alle Hindernisse o_i die das Rechteck R scheiden, wird nun ein Engstellentest durchgeführt. Dieser lässt sich algorithmisch folgendermaßen beschreiben.

function abstand_und_schittpunkt: boolean;

var $o_i \in MAP$

var $o_j \in MAP$

var w

var $l_{min} \in Q$ kürzeste Verbindungsstrecke zwischen (o_i,o_j)

abstand_und_schittpunkt:=l(l_{min})<m_B and s(w,l_{min})

function engstellenstest: boolean;

in W, MAP;

var $q_s \in Q$

var $R_q \subseteq MAP$

var $w \in W$

var $o_{(i,j)} \in R_q \, x \, R_q \, mit \, o_i \neq o_j$

$R_q := MAP \cap R$

engstellenstest:=false

for all $o_{(i,j)} \in R_q$

if abstand_und_schittpunkt then

engstellenstest:=true

endif

Beispiel für Befahrbar:

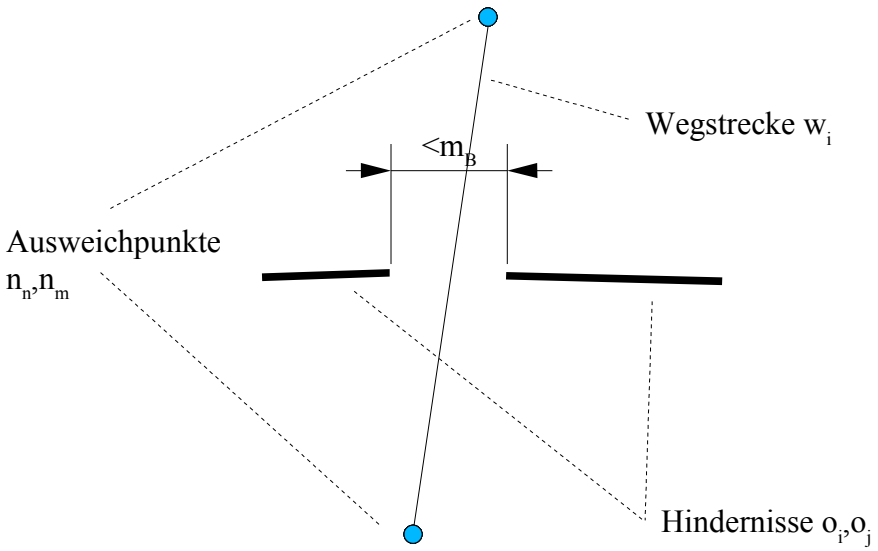


Abbildung 14: Beispiel für eine befahrbare Wegstrecke

Beispiel für nicht befahrbar:

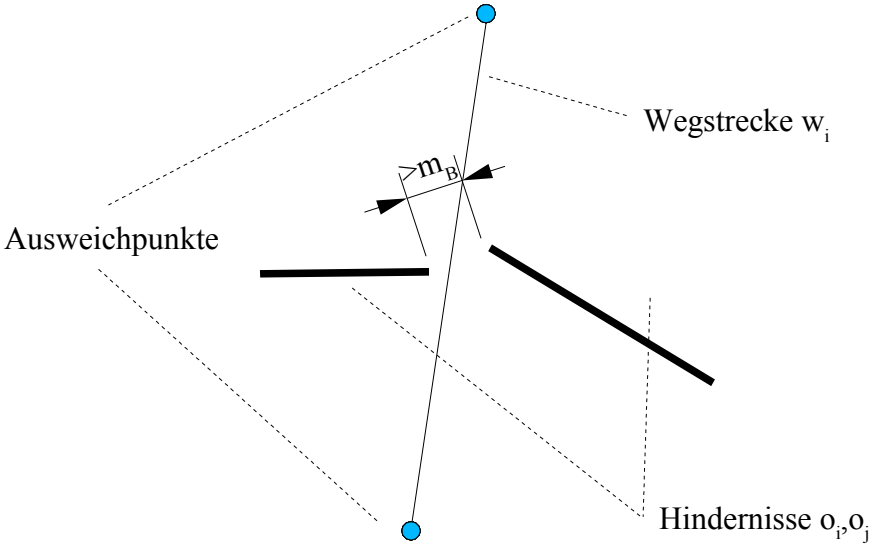


Abbildung 15: Beispiel für eine nicht befahrbare Wegstrecke

Wird eine Stelle gefunden, die durch ein Hindernis verschlossen ist, muss der getestete Weg als nicht befahrbar markiert werden und die Pfadfinung bei der geometrische Pfadsuche neu gestartet werden.

5 Bubbles zur Erstellung eines Pfades entlang einer Wegstrecke w_i

5.1 Bubbles und konvexe Polygone

Konvexe Polygone haben eine Eigenschaft, die sie ideal für das Finden von befahrbaren Wegstrecken macht. Jeder Punkt $p_i \in POLY$ innerhalb des konvexen Polygonezuges lässt sich über eine Strecken q mit jeden beliebigen Punkt $p_j \in POLY$ innerhalb des Polygonezuges verbinden ohne dass dadurch die Strecke q den Polygonezug schneidet (Abbildung 16).

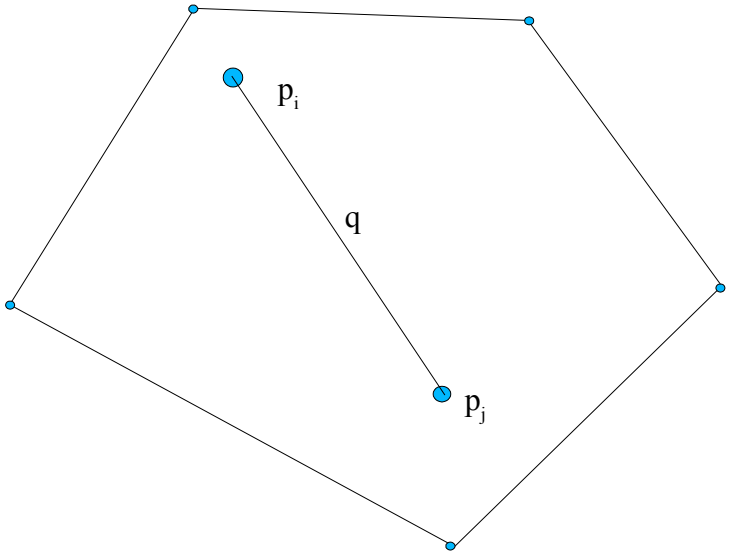


Abbildung 16: Beispiel für konvexes Polygone

Wenn wir innerhalb eines Polygone einen Rand R_A mit dem dem Abstand $m_B/2$ definieren, erhalten wir eine Untermenge von Punkten mit der auch unser AMR in der Lage ist, von jedem Punkt $p_n \in POLY$, $p_n \notin R_A$ über die Strecke q(p_n,p_m) den Punkt $p_m \in POLY$, $p_m \notin R_A$ anzufahren ohne dabei das Polygone zu berühren.

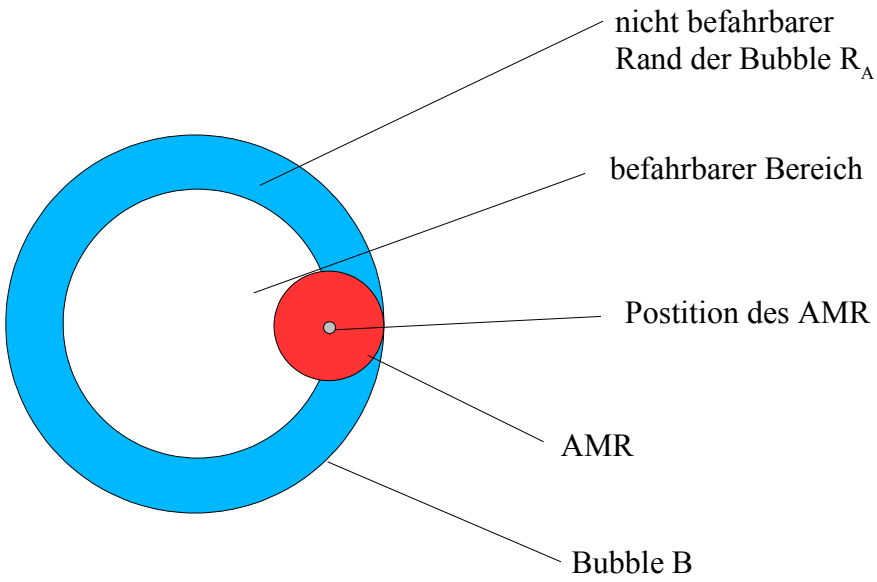


Abbildung 17: befahrbare und nicht befahrbare Bereiche innerhalb einer Bubble

Um das Arbeiten mit konvexen Polygonen möglichst einfach zu gestalten haben wir einen Sonderfall eines konvexen Polygone verwendet, einen Kreis. Er besitzt wie alle konvexen Polygone die oben genannte Eigenschaft, lässt sich aber einfacher beschreiben (r ... Radius, MP_i ... Mittelpunkt) und auch die Schnittfläche bzw. die Schnittpunkte des Randes zweier solcher Polygone ist einfach zu handhaben. Im Weiteren werden wir immer wenn konvexe Kreise gemeint sind, von Bubbles sprechen; so, wie diese auch in der Literatur in Zusammenhang mit AMR häufig genannt werden.

5.2 Wegführung zwischen 2 sich schneidenden Bubbles

Die erste Frage ist, welche Bereiche einer Bubble nun für einen AMR befahrbar werden, wenn sich zwei Bubbles schneiden. Wie aus 5.1 ersichtlich, gilt auch weiterhin das das Innere der Bubbles ohne R_A der beiden Bubbles B_1 und B_2 für den AMR weiterhin befahrbar ist. Dadurch, dass sich die beiden Bubbles überlappen, entsteht ein Gebilde wie in Abbildung 18

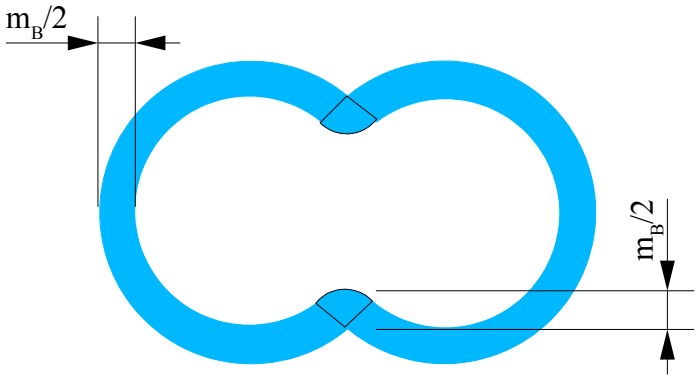


Abbildung 18: 2 sich schneidende Bubbles und der entstandene Freiraum für den AMR

Da es sich bei dem in Abbildung 18 um keinen konvexen Raum mehr handelt, wird für die Wegführung von B_1 nach B_2 eine Strecke ausgewählt die immer befahrbar ist. Eine solche Strecke

ist beispielsweise die Verbindungsstrecke zwischen den beiden Mittelpunkten MP_1 und MP_2. der beiden Bubbles.

Diese befahrbare Verbindung existiert immer dann, wenn der Abstand zwischen den beiden Schnittpunkten der beiden Bubbles > m_B ist, wie auch aus Abbildung 18 leicht ersichtlich.

Indem man also eine Folge von Bubbles entlang eines Weges konstruiert die die gerade genannte Eigenschaft aufweisen, ergibt sich durch das Verbinden der Mittelpunkte ein Pfad, der von Hindernissen frei ist. Im Folgenden sprechen wir immer dann von einem Pfad bzw. von einer Pfadführung, wenn diese unter Berücksichtigung von m_B des AMR hindernisfrei ist. Hingegen sprechen wir von einem Weg bzw. von einer Wegführung wenn diese(r) noch Hindernisse enthalten kann.

Beispiele:

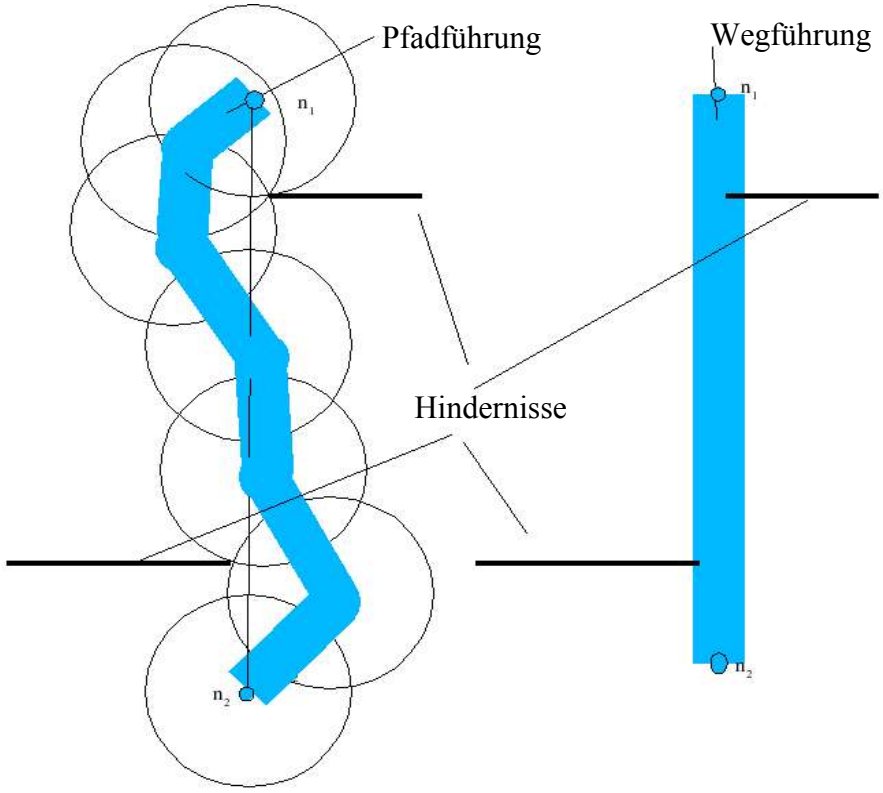


Abbildung 19: Gegenüberstellung Wegführung/Pfadführung

5.3 Erstellung eines Pfades entlang einer Wegführung S nach G

Eine Wegstreckenführung w_1,w_2 wurde wie in 4.6 auf ihre mögliche Befahrbarkeit hin untersucht. Daraus soll jetzt mit Hilfe von Bubbles ein Pfad entstehen, der es dem AMR ermöglicht, ohne Kollision mit einem Hindernis diesen abzufahren. Dazu werden entlang der Wegstrecke im Abstand $\leq m_B/2$ Ausgangspunkte für die Bubblekonstruktion erstellt. Der Wert $\leq m_B/2$ ist dabei abhängig von der Größe und dem Abstand zweier sich scheidender Bubbles. Da der Bubbble Durchmesser r_B immer größer als der Durchmesser des AMR m_B sein muß, ist es eine notwendige aber nicht hinreichende Bedingung, dass der AMR sich zwischen zwei Bubbles in einem hindernisfreiem Raum bewegt. Da der Abstand zwischen den zwei Bubbles aber variiert, kann es vorkommen, dass die Schnittpunkte einen Abstand kleiner als m_B haben und die Hindernisfreiheit nicht mehr garantiert ist. Das kommt daher, daß die Bubbles normal zur Wegführung so verschoben werden und ihr Durchmesser an den Freiraum angepasst wird d.h. der Durchmesser wird soweit verkleinert, dass die Bubble kein Hindernis mehr enthält. In Abbildung 21 wird gezeigt wie die

Bubbles verschoben werden, ohne dass sich der Radius der Bubble verändert.

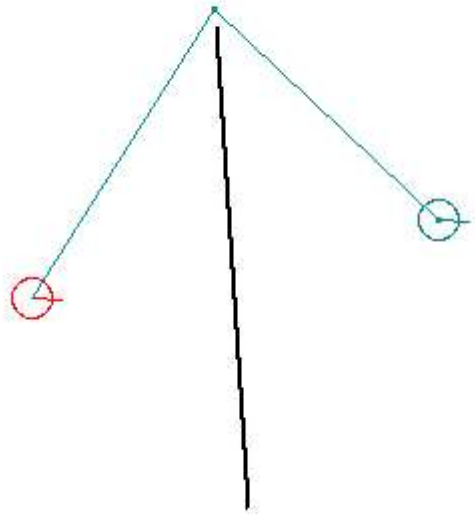


Abbildung 20: Wegfürhung um ein Hindernis herum

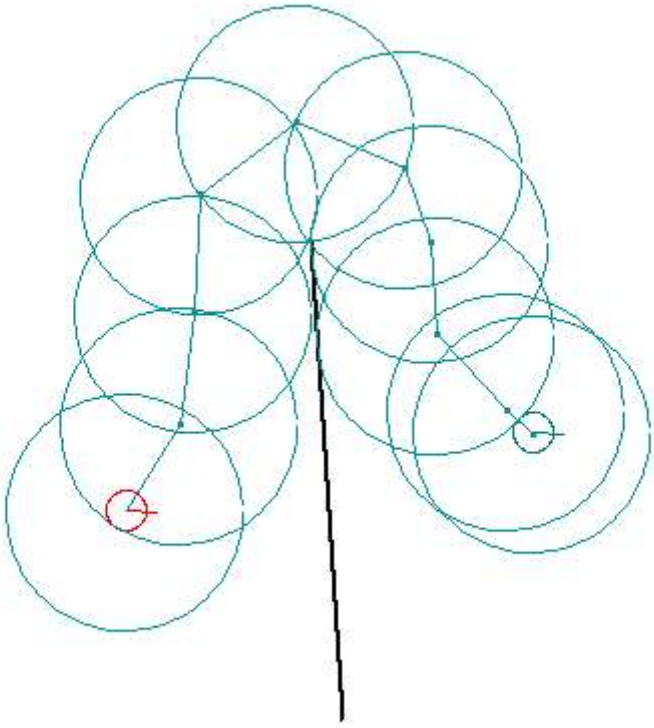


Abbildung 21: Verschieben der Bubbles, so dass sie das Hindernis nicht mehr schneiden

In Abbildung 22 hingegen werden die Bubbles nicht nur verschoben sondern es wird auch der Durchmesser so verringert, dass die Bubbles keine Hindernisse mehr enthalten.

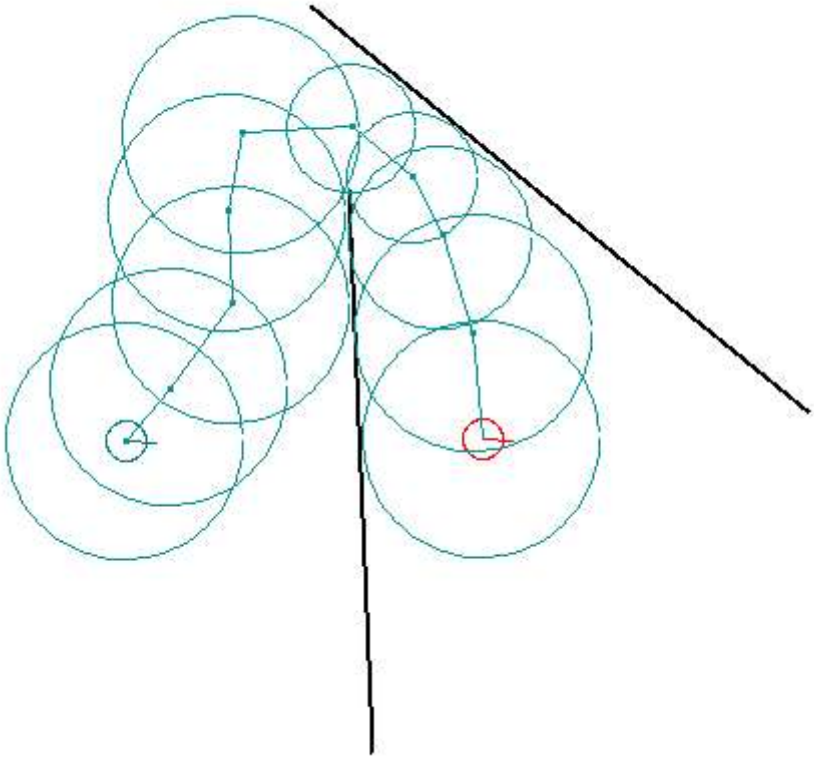


Abbildung 22: Bubblekonstrunktion mit variabler Größe und Verschiebung

Durch genügend große Bubbles kann erreicht werden, dass bei einer späteren Navigation z.B. ein entgegenkommender AMR innerhalb einer Bubble ausweichen kann, ohne dass eine komplette Neuberechnung des Pfades von Nöten ist. Das gilt, wenn andere unerwartete Hindernisse auftreten, die die Bubble nur teilweise schneiden.

5.4 Konstruktion einer Bubble

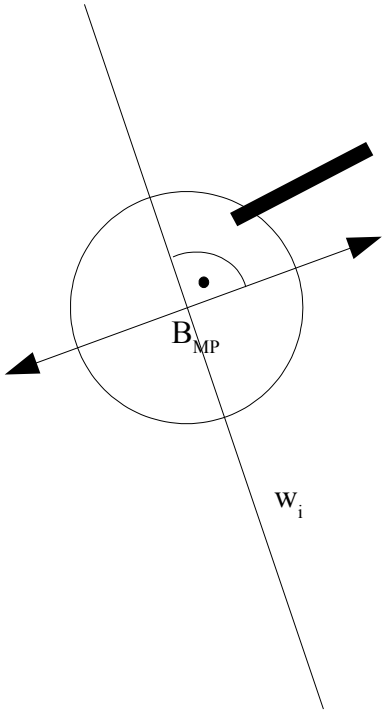


Abbildung 23: Bubblekonstruktion

Bei der Konstruktion einer Bubble wir ausgehend von einem Punkt B_{MP} auf der Wegstrecke w_i und einem Anfangsradius r_{Bmax} die Bubble B so verschoben und oder verkleinert, dass diese kein Hindernis mehr enthält. Um dies zu erreichen müssen wir 4 Fälle unterschieden:

Fall 1: Es befinden sich weder links noch rechts Hindernisse, die die Bubble schneiden. Dann sind wir fertig und die Bubble bleibt auf ihrer Position.

Fall 2: Es befinden sich nur rechts von w_i Hindernisse, die die Bubble scheiden. Dann verschiebe die Bubble genau soweit nach Links, dass sie gerade kein Hindernis mehr auf der rechten Seite schneidet. Überprüfe ob die Bubble jetzt auf Hindernisse die links von w_i liegen, scheidet und wenn ja, reduziere den Bubbledurchmesser und beginne den Test von Neuem, wenn nein, sind wir fertig.

Fall 3: Es befinden sich nur links von w_i Hindernisse, die die Bubble scheiden. Dann verschiebe die Bubble genau soweit nach Rechts, dass sie gerade kein Hindernis mehr auf der linken Seite schneidet. Überprüfe ob die Bubble jetzt auf Hindernisse, die rechts von w_i liegen, scheidet und wenn ja, reduziere den Bubbledurchmesser und beginne den Test von Neuem, wenn nein, sind wir fertig.

Fall 4: Es befinden sich links und rechs von w_i Hindernisse, die die Bubble B schneiden dann reduziere den Radius der Bubble und Beginne den Test von Neuem.

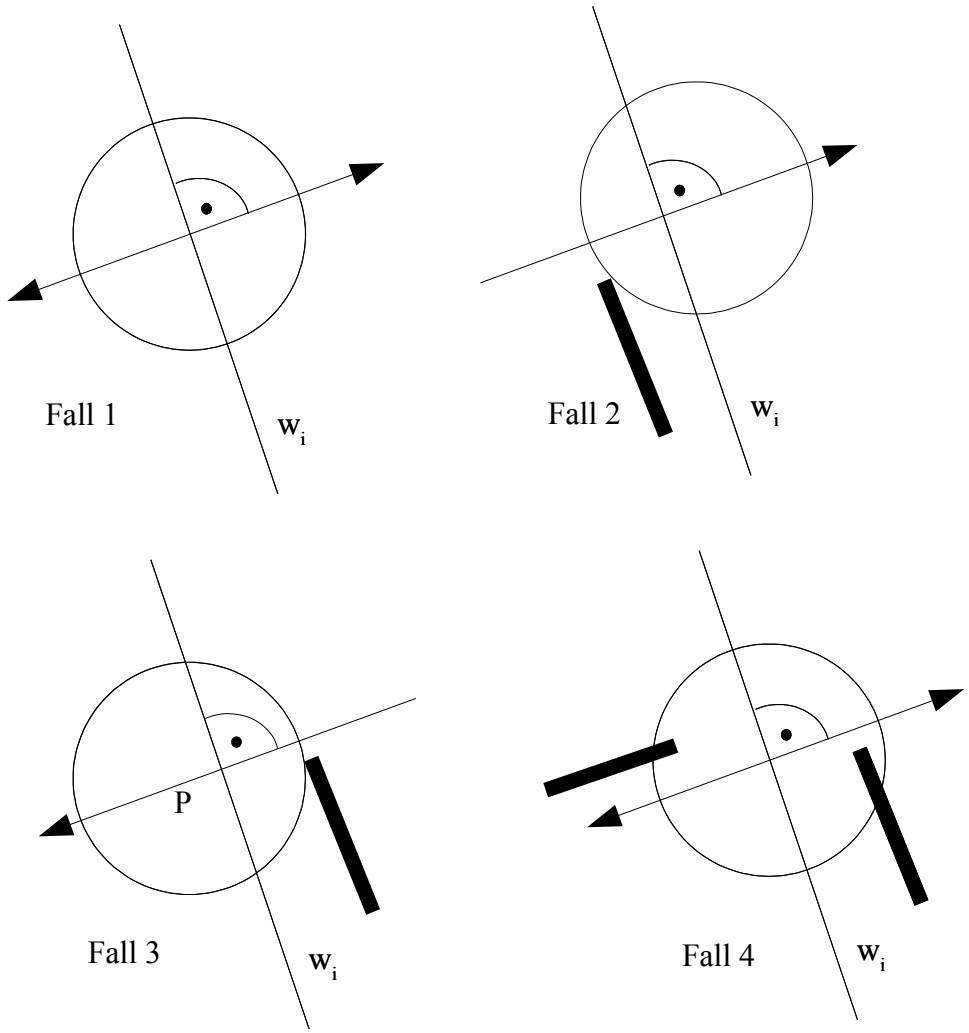


Abbildung 24: Fälle 1 - 4 beim Bubbleverschieben

Das klingt zwar jetzt recht einfach, aber wenn wir genauer hinsehen, gibt es da ein nicht zu unterschätzendes Problem. Um wieviel müssen wir den Radius reduzieren, damit wir beim nächsten Durchlauf einen Fall bekommen, bei dem die Abruchbedingung erfüllt ist? Um das zu klären schauen wir uns einige Szenarien an.
Also wenn wir zwei Hindernisse haben, eines links und eines rechts des Weges w_i, dann ist es doch einleuchtend, dass es einen Punkt und einen Radius geben muß, bei denen die Abbruchbedingung erfüllt ist. Mit etwas Rechenarbeit können wir auch auch eine Formel finden, die dann 3 Ergebnisse liefert: einmal in der Mitte zwischen den beiden Hindernissen und 2 Ergebnisse, die links neben dem rechten Hindernis und rechts neben dem linken Hindernis zum Liegen kommt. Uns interessiert aber nur das Ergebnis in der Mitte, das noch den Punkt P enthält.
Wie sieht es jetzt aber aus, wenn nicht nur zwei Hindernisse beteiligt sind. Wie man sich denken kann, kommen jetzt alle möglichen Ergebnisse zustande, bei denen die Entscheidung immer schwieriger wird, welches nun die Bubble ist, die wir suchen. Und es wird um so schwieriger, je mehr Hindernisse beteiligt sind.

Also neuer Versuch, man könnte sehr kleine Schritte machen und den Radius nach und nach soweit reduzieren bis einer der Fälle 1 bis 3 passt d.h. die Bubble enthält P und der Radius ist möglichst groß. Das Problem dabei ist nur, dass damit ein sehr großer Rechenaufwand für ein nur relativ ungenaues Ergebnis betrieben werden muss (Genauigkeit enspricht der Schrittbreite).

Noch ein Versuch, wir werden den Radius einfach mit der bewährten Methode des Zahlenratens erraten. Also unser Radius liegt zwischen R_{max} und $R_{min} = r_B$ wir beginnen mit $R_{test}=(R_{max}+R_{min})/2$,

$R_h = (R_{max} - R_{min})/2$ und machen dann mit

$R_{test} = R_{test} \pm R_h/2$ weiter, dann mit

$R_{test} = R_{test} \pm R_h/4$, dann mit

$R_{test} = R_{test} \pm R_h/8$, usw.

bis zu $R_{test} = R_{test} \pm R_h/2^{ITER}$, einer Iterationstufe ITER, die die gewünschte Genauigkeit $\delta \geq R_h * \ln(ITER)/\ln(2)$ hat. Ob wir den Wert des Radiuses R_{test} vergrößern oder verkleinern, hängt davon ab, ob die Bubble hindernisfrei plaziert werden kann Plus (in den Fällen 1-3) oder Minius (in den Fällen 2-4) wenn nicht.

Warum bekommen wir dabei eine Bubble die passt und nicht einen der vielen Fälle aus unserem ersten Versuch zur Berechnung der Bubble, der auch unbrauchbare Bubbles liefete? Die Antwort ist ganz einfach, da wir beim Verschieben der Bubble immer nur in eine Richtung schieben (Fall 2 und Fall 3) und so nie in die Verlegenheit kommen, entscheiden zu müssen, ob wir zu weit rechts oder zu weit links sind.

5.5 Bubble Verschiebealgorithmus

Definition der Bubble B:

B ist ein Tupel (B_{MP},r_B) mit $B_{MP} \in P \; und \; r_B \in \mathbb{R}$

Schittfunktion $f_B(o_i) \; mit \; o_i \in MAP$ ist true wenn o_i B schneidet, sonst false.

Positionsfunktion $f_{wi}(w_i, MAP) \; mit \; w_i \in W \; und \; o_i \in MAP$ ist true wenn o_i rechts von w_i liegt, sonst false.

$MAP_L \subset MAP \; mit \; f_{wi}(w_i, o_i) = false \wedge f_B(o_i) = true$

$MAP_R \subset MAP \; mit \; f_{wi}(w_i, o_j) = true \wedge f_B(o_j) = true$

function constructBubble

INBUT r_{Bmax}, r_{Bmin}, P,w_i, ITER,B

OUTPUT B

var r_{test}:=(r_{Bmax}+r_{Bmin})/2

for k:=1 to ITER do

begin

if $(f_B(o_i) mit\ o_i \in MAP_L) \wedge (f_B(o_j) mit\ o_j \in MAP_R) then$

$r_{test} = r_{test} - (r_{Bmax} - r_{Bmin})/(2^k)$

elseif ($f_B(o_i) mit\ o_i \in MAP_L$) then

verschiebe B noch rechts bis $f_B(o_i)$=false

if $f_B(o_i) mit\ o_i \in MAP_R$ then

$r_{test} = r_{test} - (r_{Bmax} - r_{Bmin})/(2^k)$

else

$r_{test} = r_{test} + (r_{Bmax} - r_{Bmin})/(2^k)$

endif

elseif ($f_B(o_j) mit\ o_j \in MAP_R$) then

verschiebe B noch links bis $f_B(o_j)$=false

if $f_B(o_j) mit\ o_j \in MAP_L$ then

$r_{test} = r_{test} - (r_{Bmax} - r_{Bmin})/(2^k)$

else

$r_{test} = r_{test} + (r_{Bmax} - r_{Bmin})/(2^k)$

endif

else

$r_{test} = r_{test} + (r_{Bmax} - r_{Bmin})/(2^k)$

endif

end

return B

6 Topologische Aufteilung des geometrischen Raumes

Um eine effizientere Berechnung zu ermöglichen, wird die MAP in kleinere Teilräume unterteilt, die über Türen miteinander verbunden sind. Dadurch wird verhindert das bei jedem Einfügen von Hindernissen alle Wegpunkte neu berechnetet werden. Dazu wird die MAP in kleinere Räume (kleinere Maps) unterteilt und über Türen miteinander verbunden. Daraus ergibt sich dann ein Graph mit den Türen als Knoten und den Pfaden zwischen den Türen innerhalb eines Raumes als Kanten. Zudem ergibt sich die Möglichkeit, durch eine Gewichtung der Räume einen zusätzlichen Lenkungseffekt zu erzielen, um bestimmte Gebiete (Räume) für eine Pfadführung interessant oder uninteressant zu machen. Das könnte dann wie in Abbildung 25 aussehen.

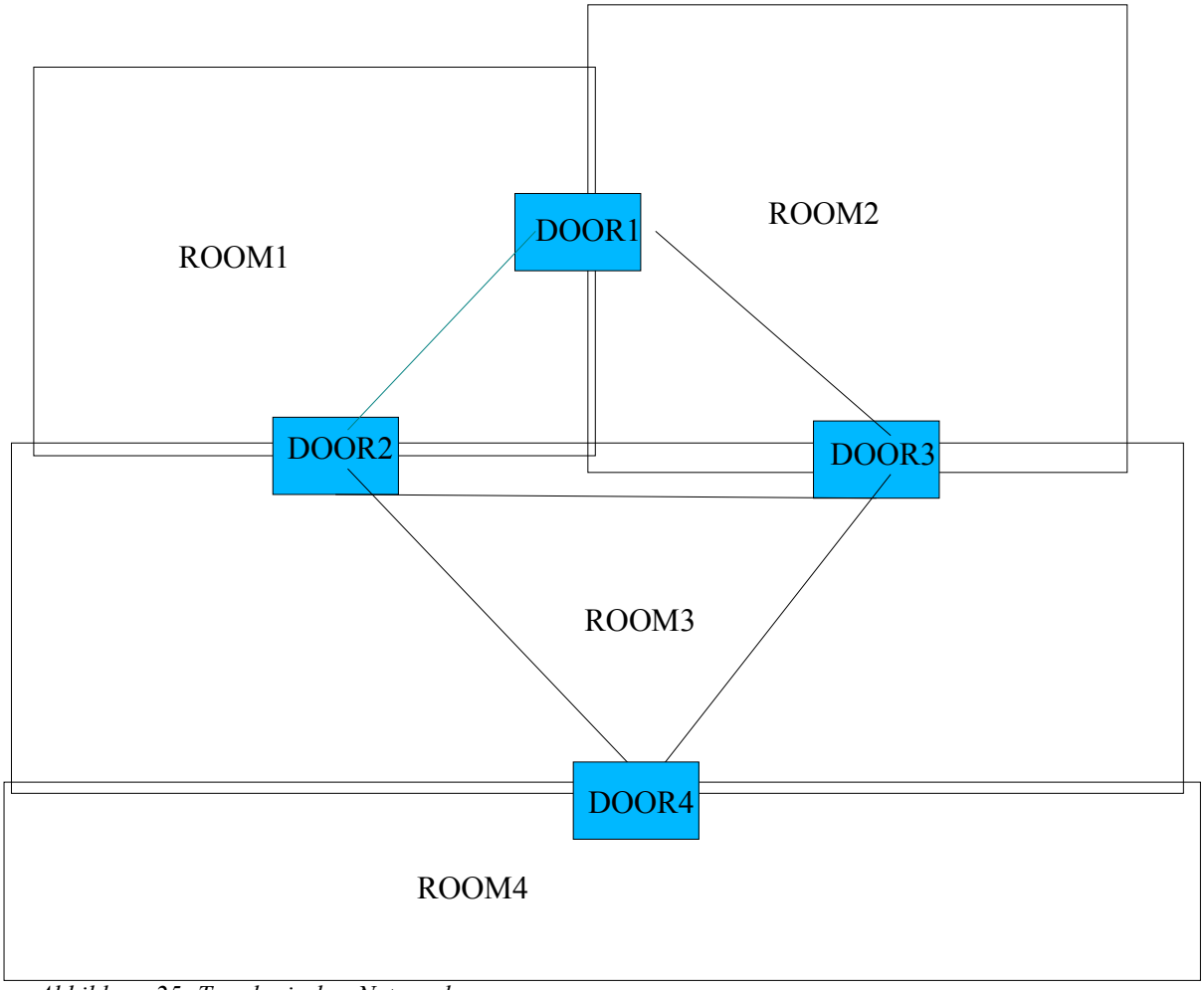


Abbildung 25: Topologisches Netzwerk

Wie in Abbildung 25 ist es dabei völlig egal ob sich die Räume überlappen , da bei einer Tür nur wichtig ist, welche Räume sie verbindet und bei den Kanten nur interessant ist, ob ein solcher Pfad existiert. Räume dienen in der Abbildung nur zur Veranschaulichung wie das topologische Netzwerk aufgebaut ist.

6.1 Zusammenhang zwischen topologischem Netzwerk und den MAPS

Jedem der Räume im Topologischen Netzwerk ist eine Geometrische MAP zugeordnet, deren Methoden einen Pfad liefern, der aus einer Folge von Bubbles zusammengesetzt ist. Die Türen sind im Topologischen Netzwerk Knoten mit einem Türmittelpunkt $DOOR_{Pi}$, einem Tür Radius $DOOR_{rT}$, einer Türrichtung $DOOR_{PHI}$, den beiden Raumnummern ($DOOR_{NR1}$, $DOOR_{NR2}$) den Kosten $DOOR_{costS}$. für die Pfadführung vom Startpunkt S bis zu dieser Tür und den Kosten

$DOOR_{costG}$. für die Pfadführung von der Tür bis zum Zielpunkt G.

Dabei ist es für die Tür eine notwendige Bedingung, dass sie in den beiden MAPS der Topologischen Räume die sie verbindet, im Umkreis einer Bubble mit Radius $DOOR_{rT}$ und Mittelpunkt $DOOR_{Pi}$ hindernisfrei ist. Mit Hilfe der Richtung der Tür $DOOR_{PHI}$ wird die Tür in beiden MAPS mit je 3 Hindernissen versiegelt was später dazu führt das kein Pfad gefunden wird, der die Tür von der falschen Seite anvisiert. Siehe Abbildung 26.

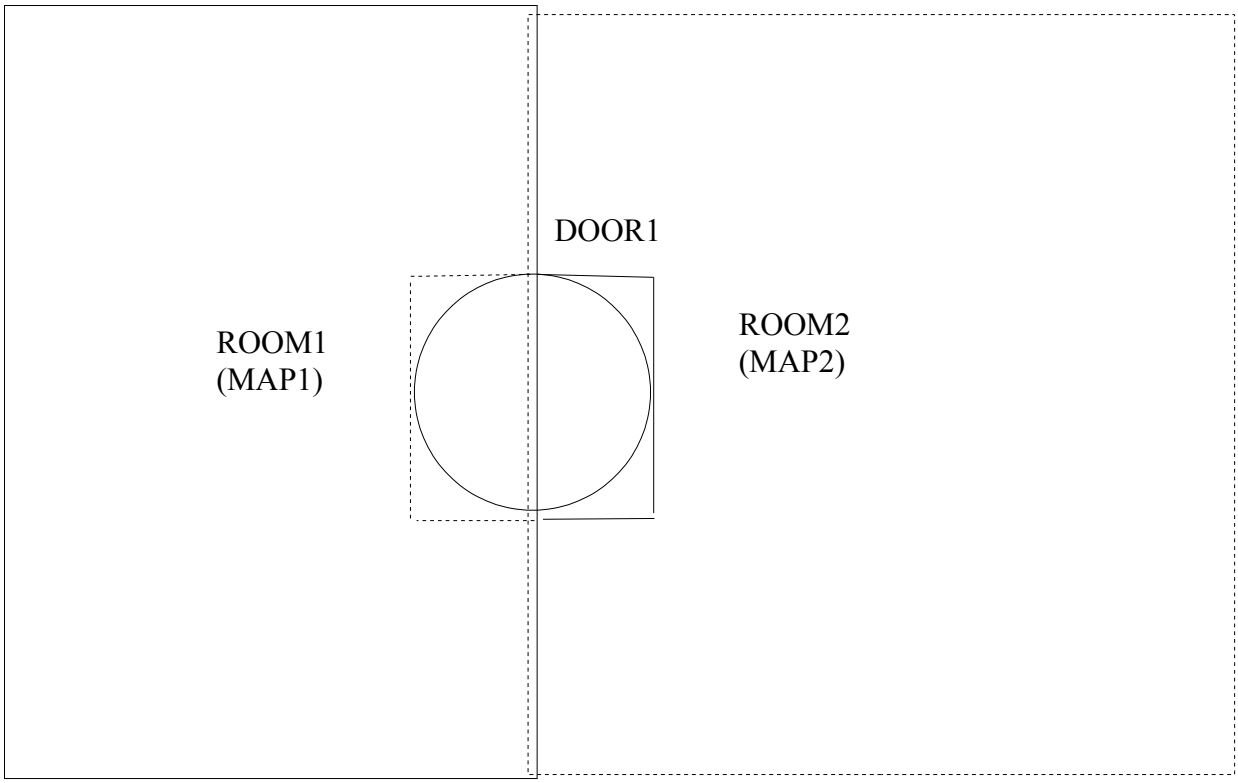


Abbildung 26: Zusammenhang zwischen MAPS und Rooms

6.2 Pfadsuche im topologischen Netzwerk

Bei der Pfadsuche im topologischen Netzwerk wird ausgehend vom Startpunkt ein Pfad zu allen Türen im selben Raum berechnet. Die Türen werden in einen sortierten Binärbaum anhand der geschätzten Kosten bis zum Ziel $DOOR_{costG}$ eingetragen. Aus diesem Binärbaum wird der beste Kandidat ausgewählt und die Prozedur wird solange wiederholt, bis das Ziel erreicht wird oder es keine zu .testenden Kandiaten mehr gibt (A*Algorithmus). Aus den Werten von $DOOR_{costS}$ der Türen beginnend mit dem Ziel G läßt sich dann der gefundene Weg durch Ermitteln der jeweils besten Werte $DOOR_{costS}$ der benachbarten Türen ermitteln.

6.2.1 Abschätzung der Kosten bis zum Ziel

Die Abschätzung der Kosten von einer Tür zum Ziel ergibt sich aus der Kosten vom Start S bist zur der Tür $DOOR_i$ addiert mit dem Abstand $DOOR_i$ und G mal einem Korrekturfaktor k_{TG} für die Abweichung zwischen den Kosten des Pfades der Luftlinie und der wirklichen Pfadführung. Es hat sich ein Wert zwischen 1.2 und 2 bei den Testläufen als realistisch erwiesen.

6.2.2 Algorithmus für die Kostenabschätzung von einer Tür bis zum Ziel

function f_{TCost}

INPUT $DOOR_i$,G, k_{TG}

OUTPUT Cost

cost := $DOOR_{costS}$+distance ($DOOR_i$, G)*k_{TG}

7 Datenstrukturen und Details der Realisierung

7.1 Realisierung

Die Simulation wurde unter OS Linux in der Programiersprache C++ und mit Hilfe des QT Toolkit zur Erzeugung der grafischen Oberfläche realisiert. Als Programmierwekrzeug haben wir Kdevelop verwendet. Die Simulationsumgebung besteht im Wesentlichen aus der Oberfläche zur Eingabe und Steuerung, dem toplogischen Netzwerk das der Oberfläche die Daten liefert und den geometrischen MAPS die in den Räumen im topolgischen Netz gespeichert sind.

7.2 Aufbau des topologischen Netzwerks

Das topologische Netzwerk besteht aus einer Liste von Räumen die durch Türen miteinander verbunden sind: Jede Tür in einem Raum ist mit allen anderen Türen im Raum über Connections verbunden. Daraus ergibt sich ein Netzwerk mit Türen als Knoten und Connections als Kanten.

7.2.1 Die Türen

Die Türen verbinden zwei Räume und sind durch folgende Variablen definiert:

room *room1;	pointer auf den ersten Raum (enthält die Raumnummer und MAP)
room *room2;	pointer auf den zweiten Raum (enthält die Raumnummer und MAP)
long d_nr;	eindeutigen Nummer zur Identifikation einer Tür
point d_p	Mittelpunkt für die Tür Bubble;
double to_start;	Gewichtung des Pfades von Startpunkt bis zu dieser Tür
double s_to_goal;	Schätzwert der Gewichtung des Pfades von Startpunkt bis zum Ziel über dieses Tor
room *next_room;	enthält den Raum der in Pfadrichtung (S -> G) liegt
double direction;	gibt die Richtung an, in der der zweite Raum liegt (für das Zeichnen wichtig)
double width;	Breite der Tür

7.2.2 Die Connections

Die Connections verbinden die Türen in einem Raum über Pfade und sind durch die folgenden Variablen definiert:

door *s_door;	pointerStarttür für eine Connection
door *g_door;	pointer auf Zieltür für eine Connection
path c_path;	enthält den Pfad für die Connection
way_status p_status;	Gibt an, ob für eine Connention eine Pfadführung möglich ist

7.2.3 Die Räume

long r_nr;	eindeutige Raumnummer
area_list *area_l;	enthält die geometrischen Daten und die Funktionen zur Berechnung eines Pfades für den Raum (MAP)
amr *r_amr;	AMR Definition für geometrische Pfaderstellung;
door_list d_list;	Liste alle Türen im Raum

connection_list c_list;	Liste aller Verbindungen im Raum
obs_list o_list;	Liste enthält alle Hindernisse, die in diesen Raum eine Rolle spielen
point bl;	unterster linker Punkt des Raumes (der MAP)
point tr;	oberster rechter Punkt des Raumes (der MAP)
bool connection_status;	gibt an ob die Pfade schon berechnet wurden

7.3 Aufbau einer MAP (geometrischer Raum)

Eine MAP besteht aus einem 4 - bätterigen Baum, der den Raum in 4 Abschnitte teilt. An den Blättern des Baumes sind Listen angefügt, die folgende Bedeutung haben:

Hindernisliste: Hindernisse wie z.B. Mauern die nicht passierbar sind
Mittelpunkt des Quadrats
Wegpunkte: Wegpunkte über die die Wegplanung erfolgt.

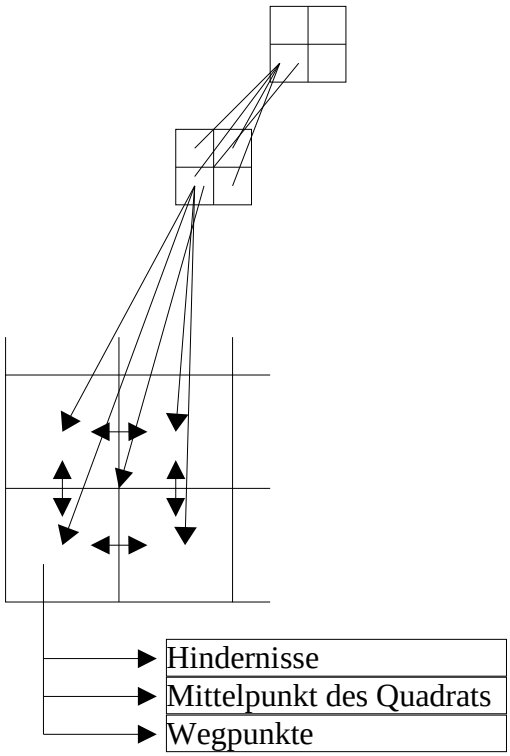


Abbildung 27: Baumstruktur einer Map

Die Blätter des Baumes werden dann gegenseitig wie in Abbildung 27 verlinkt; so, dass die Suche von Hindernissen in der Umgebung eines Blattes ein erneutes Durchlaufen der Baumstruktur unnötig macht. Damit kein offener Rand ensteht, wird die MAP um einen blattbreiten Rand erweitert, der keine Hindenisse enthält. Zusätzlich wird die Außenseite des Rands mit sich selbst verlinkt; so, dass der Rand beim Durchsuchen der Umgebung als unendlicher leerer Raum wirkt.

7.3.1 Einfügen von Hindernissen und Wegpunkten in eine MAP

Jedes Blatt der MAP enthält eine Liste von Links auf die Hindernisse, die die Blattfläche schneiden.

Abbildung 28. Um die Anzahl der Einträge zu begrenzen wird beim Erzeugen der MAP für die Blattbeite ein Durchschnittswert der Länge der Hindernisse in der MAP berechnet. Dadurch kommen im schlechtesten Fall duchschnittlich 3 Eintäge pro Hindernis zustande. Das Eintragen der Wegpunkte erfolgt in einer eigenen Liste in den Blättern.

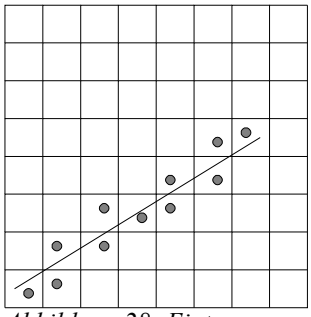


Abbildung 28: Eintragen von Hindenissen in die Map

7.3.2 Hindernisse einfügen offline

Zu Beginn werden alle bekannten Hindernisse in die Baumstruktur eingetragen . Dabei braucht man noch keine Ausweichpunkte berücksichtigen. Sobald alle Hindernisse in die MAP eingetragen sind werden die Ausweichpunkte konstruiert.

7.3.3 Hindernisse einfügen online

Wenn Hindernisse existieren und die Ausweichpunkte bereits konstruiert wurden, hat das Einfügen eines Hindernisses Folgen auf die bestehenden Ausweichpunkte und auf die bestehende Wegführung. Deshalb müssen Ausweichpunkte, die zu nah an dem eingefügten Hindernis liegen, neu berechnet und die Wegführung auf eventuelle Inkonsistenz überprüft werden. Also muß jeder Ausweichpunkt darauf überprüft werden, ob er sich im Umgebungsradius m_r des neu eingefügten Hindernisses befindet.

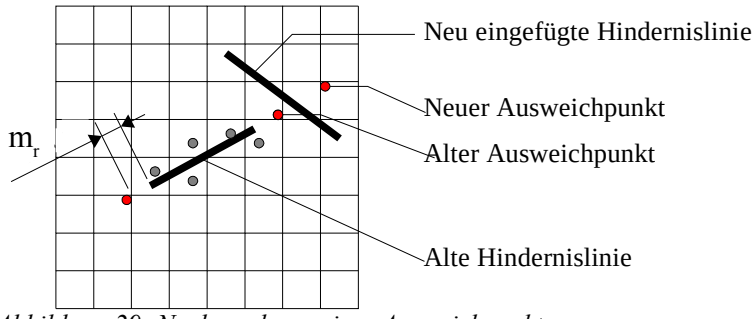


Abbildung 29: Neuberechnug eines Ausweichpunktes

7.3.4 Hindernisse entfernen online

Genau wie beim Einfügen hat das Entfernen von Hindernissen Auswirkungen auf Ausweichpunkte und die Wegführung. Auch hier müssen mögliche Inkonsistenzen gelöst werden.
Wird also eine Hindernisstrecke entfernt, so müssen alle Ausweichpunkte, die durch die Hindernislinie beeinflusst wurden, neu konstruiert werden. Das Problem dabei ist, dass man im Nachhinein nicht mehr feststellen kann, welche Hindernisse die Wegpunktkonstruktion beeinflusst haben. Die einfachste Lösung ist die Neuberechnung aller Wegpunkte in der Ensprechenden MAP,

was aber einen erheblichen Rechenaufwand erfordert.

7.4 das Wegenetz in der MAP

Das Wegenetz WN wie unter 4.1 ist in der MAP implementiert. Dazu werden Datenstukturen für die Wegpunkte und Wege definiert.

7.4.1 Wegpunkte

Wegpunkte besitzt folgende Variablen:

double x,y;	Position des Wegpunktes
double to_start;	Gewicht für die kürzeste Wegverbindung zum Start
double to_goal;	Prognose für das Gewicht des kürzesten Weges zum Ziel unter Verwendung aller passierbaren und nicht getestenen Wege
way_list wl;	Liste aller Wege die vom Wegpunkt ausgehen
obstacle *obs	Pointer auf das Hindernis welches den Wegpunkt nötig machte

7.4.2 Wegstrecke

eine Wegstecke enthält folge Variablen:

waypoint *wp1;	erster Wegendpunkt;
waypoint *wp2;	zweiter Wegpunkt
double weight;	Gewichtung für den Aufwand die Wegstrecke abzufahren
way_status w_status;	gibt an ob eine Weg befahrbar, nicht befahrbar oder noch nicht getestet wurde

Daraus ergibt sich bei der Wegsuche eine Wegführung die in einer Liste an die Bubblekonstruktion übergeben wird. Diese wiederum gibt eine Liste von Bubbles zurück. Und letztere wird dann dem topologischen Netzwerk als Path übergeben.

8 Beschreibung der Simulations Umgebung

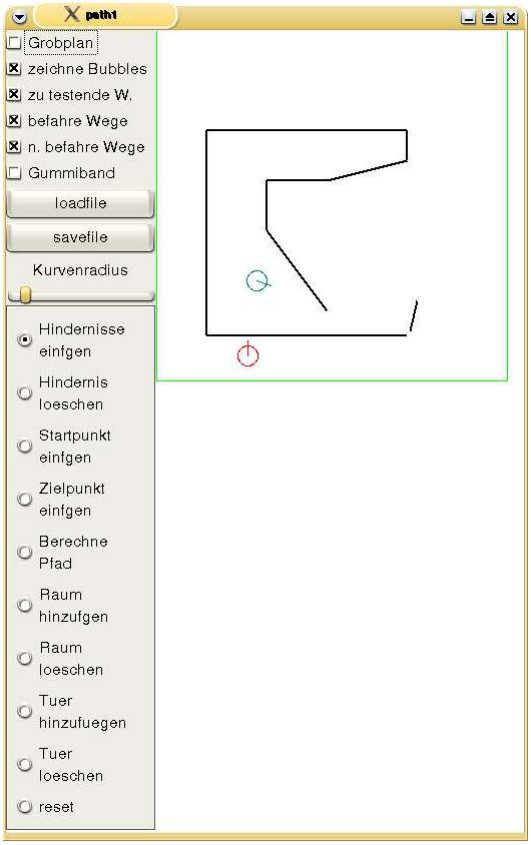


Abbildung 30: Simulationsumgebung

Die Testumgebung wie in Abbildung 30 besteht im Wesentlichen aus einem Kontrollfeld links (grauer Bereich) und der Simulationdarstellung rechts (weißer Bereich).

8.1 Beschreibung des Kontrollfeldes

Das Kontrollfeld steuert alle Funktionen der Testumgebung. Das Kontrollfeld kann in drei Bereiche unterteilt werden Darstellungssteuerung (Grobplan bis Gummiband), Speicherbereich für die Simulation (loadfile,savefile) und Bearbeitungsbereich (Kurvenradius bis reset).

- Grobplan
  Mit dieser checkbox wir die Funktion der Bubblekonstruktion ausgeschalten. Dabei werden statt der Bubble nur die Wegführung (wenn verfügbar) dargestellt.
- Zeichne Bubbles
  Mit dieser checkbox schaltet des Darstellen der Bubbles bzw. die Wegführung ein und aus. Ob die Wegführung oder die Bubbles gezeichnet werden hängt von der Grobplan ckeckbox ab.
- zu testende Wege
  Mit dieser checkbox schaltet man das Darstellen der Wegstrecken ein und aus, die beim Ausweichen eines Hindernisse erzeugt wurden aber nicht auf Hindernisfreiheit getestet wurden.
- befahrbare Wege
  Mit dieser checkbox schaltet man das Darstellen der Wegstrecken ein und aus, die beim Ausweichen eines Hindernisse erzeugt wurden und befahrbar sind.
- nicht befahrbare Wege

Mit dieser checkbox schaltet man das Darstellen der Wegstrecken ein und aus, die beim Ausweichen eines Hindernisse erzeugt wurden und als nicht befahrbar sind.

- Gummiband
  zeichnet ein Gummiband das im Abstand des Kurvenradiuses zu den Schnittpunkten der Bubbles liegt, falls möglich. Ist dies nicht möglich wird es in die Mitte zwischen die Punkte gelegt. (hat einen Fehler wenn die Schnittpunkte nicht in Fahrrichtung des AMRs liegen).
- loadfile
  Lädt eine gesicherte Simulationsumgebung.
- savefile
  Speichert die aktuelle Simulationsumgebung.
- Kurvenradius
  Ändert den Abstand beim Gummiband und die Bubblegröße.
- Hindernis einfügen
  Wird dieser ratiobutton ausgewählt kann man mit der Maus Hindernisse in die Testumgebung zeichen. Das Hinderniss wird dabei den Räumen zugeordnet die es schneiden. Wird kein Raum vom Hindernis geschnitten wir es bei der Simulation nicht berücksichtigt. Die Bestimmung der Endpunkte erfolgt durch Drücken (p1), Ziehen und Loslassen (p2) der linken Maustaste.
- Hindernis löschen
  Wird dieser ratiobutton ausgewählt kann man mit der Maus Hindernisse aus der Testumgebung löschen. Dabei wird das Hindernis gelöscht das am Nähesten beim Drücken der linken Maustaste am Klickpunkt dran ist.
- Startpunkt
  Wird dieser ratiobutton ausgewählt kann man mit der Maus den Startpunkt des AMR festlegen. Dabei drückt (p1), zieht und läßt die linke Mausetaste an der gewünschen Position los (p2). p1 bestimmt den Referenzpunkt des AMRs, p2 bestimmt die Ausrichtung des AMR bezogen auf p1.
- Zielpunkt
  Wird dieser ratiobutton ausgewählt kann man mit der Maus den Zeilpunkt des AMR festlegen. Dabei drückt (p1), zieht und läßt die linke Mausetaste an der gewünschen Position los (p2). p1 bestimmt den Referenzpunkt des AMRs, der p2 bestimmt Ausrichtung des AMR bezogen auf p1.
- Berechne
  Wird dieser ratiobutton ausgewählt, geht die Simulationsumgebung in den Simulationsmodus und berechnet einen Pfad oder Weg.
- Raum hinzfügen:
  Wird dieser ratiobutton ausgewählt so kann man in der Simulationsumgebung einen Raum zeichnen. Dabei wird durch Dücken (p1), Ziehen und Loslassen (p2) ein Rechteck parallel zur x-Achse gezeichnet. Hindernisse die beim Einfügen des Raumes innerhalb des Raumes befinden werden nicht eingefügt.
- Raum loeschen
  Wird dieser ratiobutton ausgewählt, so kann man in der Simulationsumgebung einen Raum löschen. Dabei werden die Räume gelöscht, die den Drückpunkt enthalten. Die Hindernisse werden dabei nicht gelöscht.
- Tuer einfuegen
  Wird dieser ratiobutton ausgewählt so kann man in der Simulationsumgebung eine Tür einfügen. Dies erfolgt durch Drücken (p1im ersten Raum) , Ziehen und Loslassen (p2 im zweiten Raum). Der Mittelpunkt der Tür liegt dann in der Mitte der Strecke (p1,p2) und die Ausrichtung erfolgt

normal zur Strecke (p1,p2).

- Tür löschen
  Wird dieser ratiobutton ausgewählt, so kann man in der Simulationsumgebung eine Tür löschen. Dabei wird beim Drücken der Mausetaste die am nächsten liegende Tür zum Drückpunkt gelöscht.

9 Untersuchnug des Laufzeitverhaltens

Das Laufzeitverhalten setzt sich zusamen aus:

1. Laufzeit der Grobplanung
2. Laufzeit der Verifikation und Feinplanung
3. Laufzeit für das Erstellen des Pfades mit Hilfe von Bubbles
4. Laufzeit für die Pfadfindung im Topologischen Netzwerk

9.1 Laufzeitverhalten der Grobplannung

Die Laufzeit des Algorithmus hängt von der Anzahl der zu berechnenden Wegstrecken n_w ab. Die maximale Anzahl der Wegstrecken ist wiederum von der Anzahl der Ausweichpunkte n_{wp} abhänig. Die Anzahl der Ausweichpunkte ergibt sich wiederum aus der Anzahl der Hindernisse n_o-1, zwei Ausweichpunkte pro Hindernis und dem Start- und Zielpunkt mit $2*n_o$.

Damit ergibt sich im schlechtesten Fall bei einem spät möglichsten Abbruch, das heißt alle Wegstrecken werden berechnet und verifiziert, eine Wegstreckenanzahl von:

$$n_w=(2*n_o-1)+(2*n_o-2)+(2*n_o-2)++(2*n_o-2)+...+(2*n_o-2*n_o+2)+1$$

$$n_w=\frac{((2*n_o-1)+(1))}{2}*(2*n_o-1)=2*n_o^2-n_o$$

was einer Laufzeitordnung von $O(n_o^2)$ ergibt.

Der Aufwand zu verifzieren ob eine Wegstecken hindernisfrei ist, hängt von der Länge der Wegstrecken ab die zu verifiziern sind. Um aber ein Obergrenze festzusetzen nehmen wir an, dass jedes Hindernis auf Schnittpunkte mit der Wegstrecke getestet wird, was dann für die gesamte Wegenetzverifikation $O(n_w*n_o)=O(n_o^3)$ ergibt. Dies ist auch logisch, denn der Algorithmus geht dann nach der Brute Force Methode für das Erstellen eines Visibility Graphen vor.

Das Finden des kürzesten Weges im Graphen läßte sich in $O(n_w*\log(n_w))$ erledigen, was eine Gesamtordnung von $O(n_o^3)+O(n_o^2*\log(n_o^2))=O_1(n_o^3)$ ergibt.

9.2 Laufzeitverhalten der Verifikation und Feinplanung

Da entlang einer Wegführung maximal n_o-1 Hindernisse sein können und für jedes Paar $(o_i, o_j) mit\, o_i \neq o_j$ dann eine Überprüfung auf eine Engstelle gemacht, wird ergibt sich für den Test eine maximale Ordnung:

$$O((n_o-1)*(n_o-2))=O_2(n_o^2)$$

9.3 Laufzeitverhalten für das Erstellen des Pfades mit Hilfe von Bubbles

Das Erstellen einer Bubble wird mit für eine konstante Genauigkeit immer ITER-schritte benötigt. Sollten jedesmal alle Hindernisse beteiligt sein, dann wird mit eine Weglänge l_w und einem minmalen Bubbleabstand $m_B/2$ das Ziel in $roundup((w*l_w)/m_B)=n_B$ Schritten erreicht. Das gibt dann für die Ordung $O_3(n_B*n_o)$. Dieser Wert sollte aber nicht erreicht werden, wenn die Hindernisse entlang des Weges und der MAP gleichmässig verteilt sind.

9.4 Laufzeitverhalten für die Pfadfindung im Topologischen Netzwerk

Das Erstellen eines Pfades im topolgischen Netzwerk wird der Anzahl der Türen n_D (Knoten) mit

$$O_4(n_D^2 * \log(n_D^2))$$

dem A*Algorimus eine Ordnung

9.5 Laufzeitverhalten für den gesamten Algorithmus

Aus den oben ermittelten Laufzeiten ergibt sich für die Gesamtlaufzeit eine worst case Ordnung

$$O_1(n_o^3)+O_2(n_o^2)+O_3(n_B * n_o)+O_4(n_D^2)=O_1(n_o^3)+O_3(n_B * n_o)+O_4(n_D^2 * \log(n_D^2))$$

wobei n_O die Anzahl der Hindernisse im Raum mit den meisten Hindernissen ist.

9.6 Laufzeitverhalten im besten Fall

Im besten Fall liegt der Start S und das Ziel G im gleichen und einzigem Raum und es besteht eine Sichtverbindung mit der Beite $m_{B,}$ dann ist die Verifiktion der Wegstrecken bei der Grobplanung beendet mit $O_1(1)$. Die Verifikation ergibt kein Hinderns das im Abstand m_B zu w_1 liegt, was eine $O_2(1)$.ergibt. Die Ordnug zum Berechnen der Bubbles wird $O_3(n_B)$.Es gibt im topolgischen Netzwerk genau einen Pfad (S,G) und damit ergibt sich $O_4(1)$.

Für die gesamte Laufzeit gilt für den best case also:

$$O_3(n_B)$$

9.7 Testen der worst case Laufzeiten anhand von Beispielszenarien

Zum Testen der Laufzeit verwenden wir ein Spirale daduch werden die vorgeschlagen Wege aus A* immer falsch sein. Das liegt daran, dass der kürzeste Schätzwert für einen Weg immer die Spirale schneidet. Dadurch wollen wir erreichen, daß ein möglichst vollständiges Wegenetz ensteht. Um den Anstieg der Laufzeit in Abhänigkeit der Hindernisse nachvollziehen zu können, vergrößern wir kontiuierlich die Windungen der Spirale um 1 wie es in den Abbildung 31 bis Abbildung 36 gezeigt wird. In Abbildung 37 bis Abbildung 39 hingegen wird die Anzahl der Hindernisse verdoppelt indem die Hindernissegmente aufgeteilt werden. Daraus wollen wird dann die Laufzeit Abschätzungen überprüfen.

Das Testsystem hat folgende Konfiguration:

Prozzessor: AMD DURON 1GHz

Betiebssystem: Debian Sarge LINUX

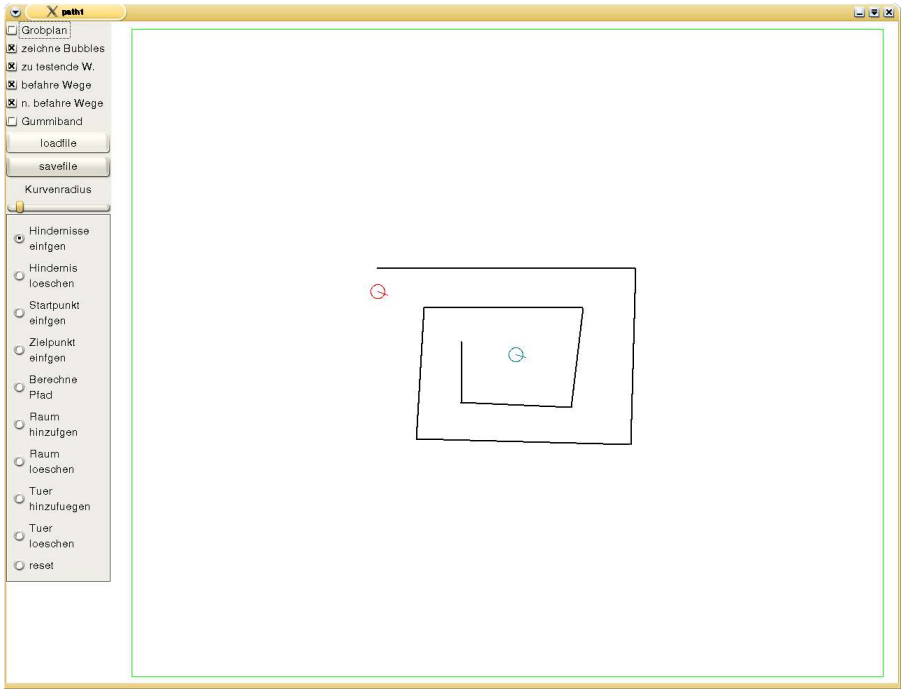


Abbildung 31: Testszenario s8

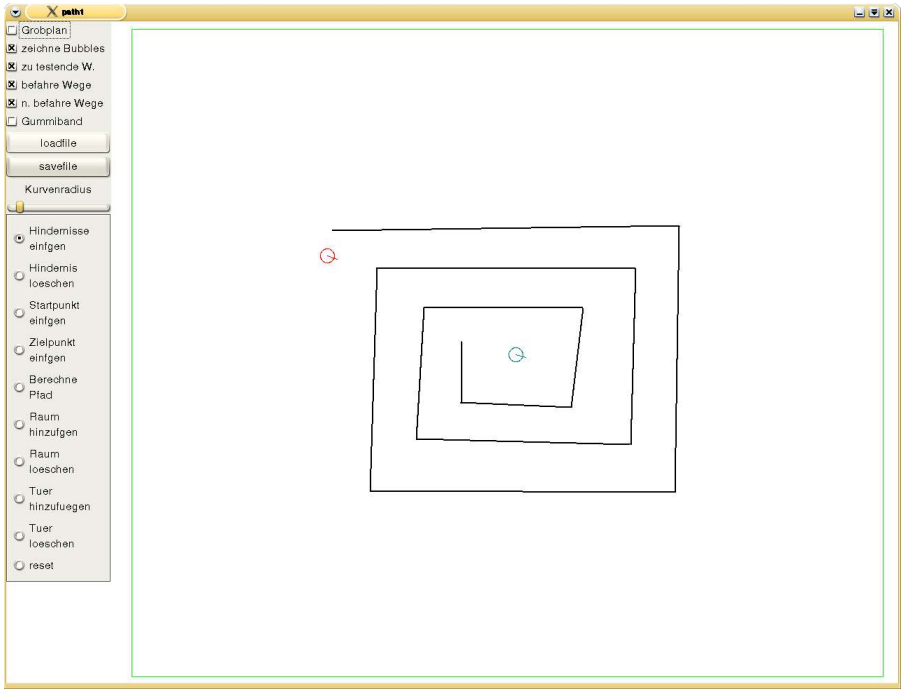


Abbildung 32: Testszenario s12

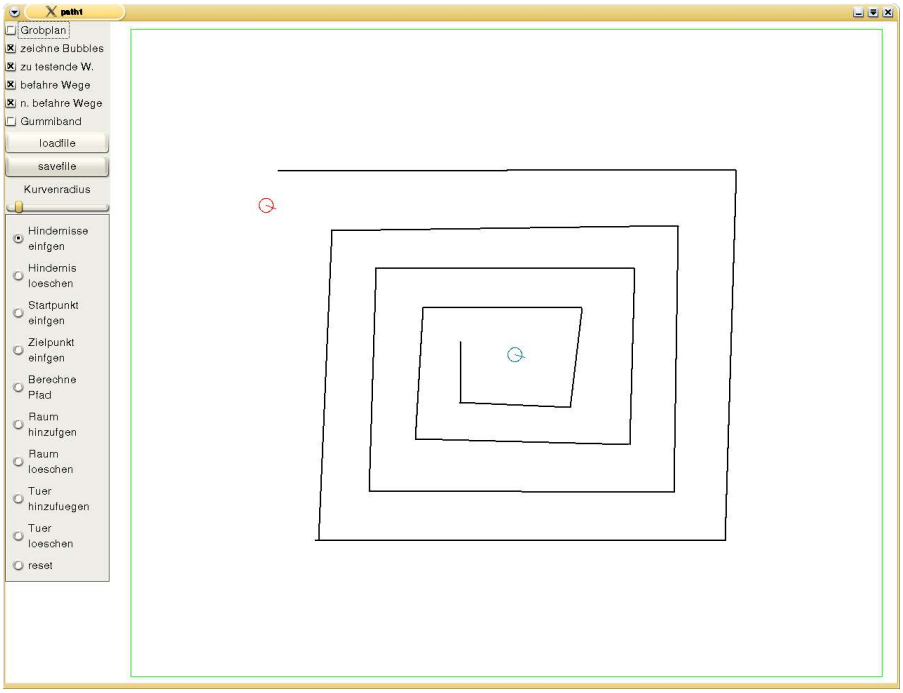


Abbildung 33: Testszenario s16

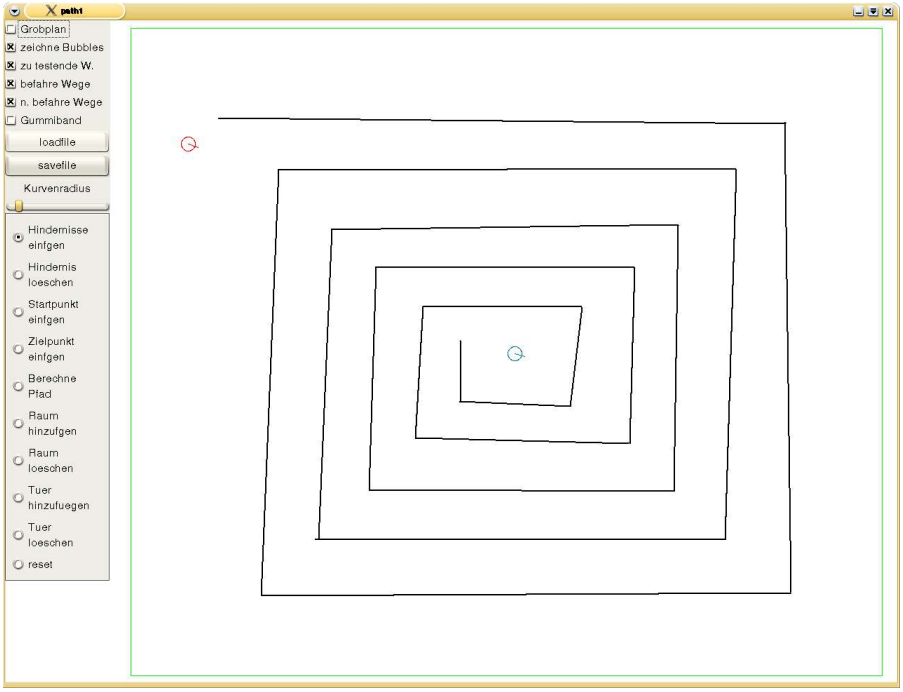


Abbildung 34: Testszenario s20

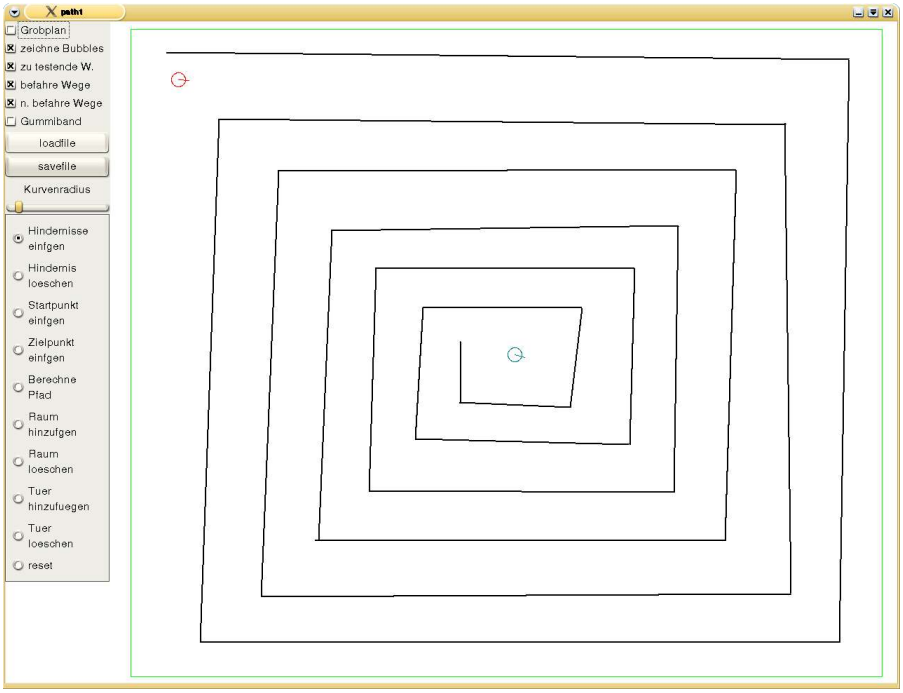


Abbildung 35Testszenario s24

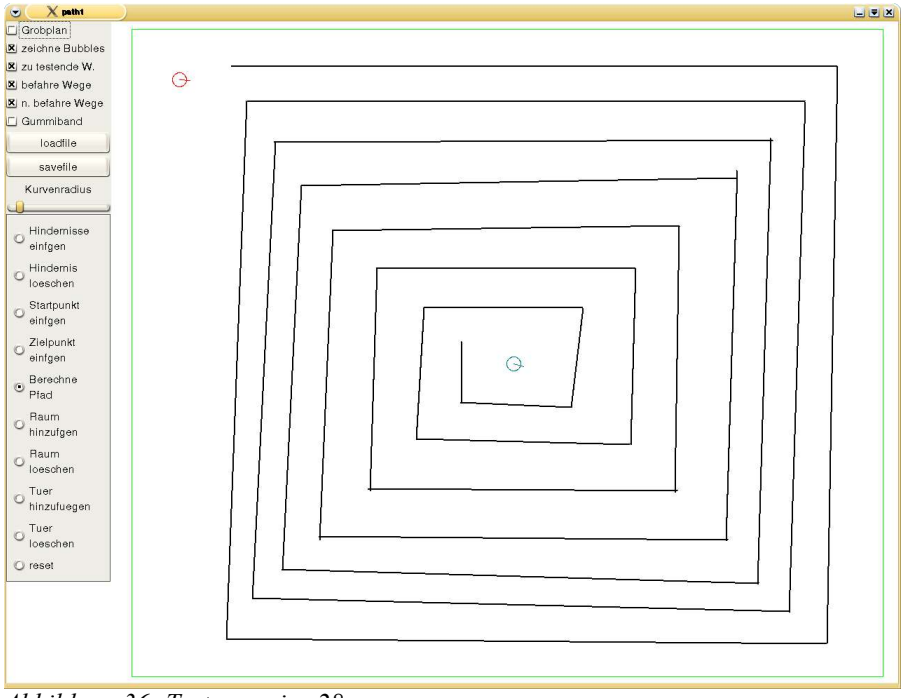


Abbildung 36: Testszenario s28

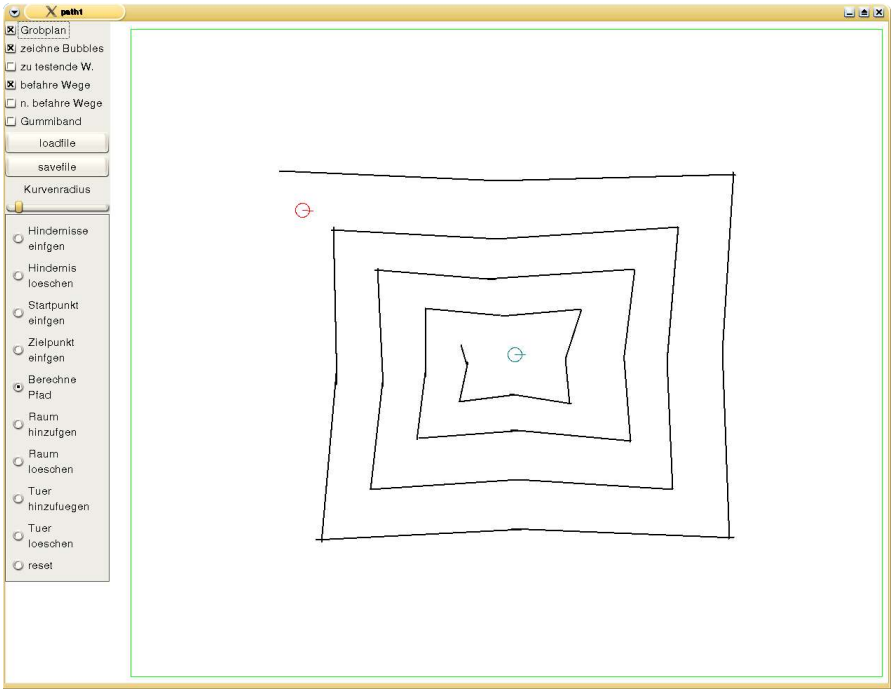


*Abbildung 37: Testszenario s16*2*

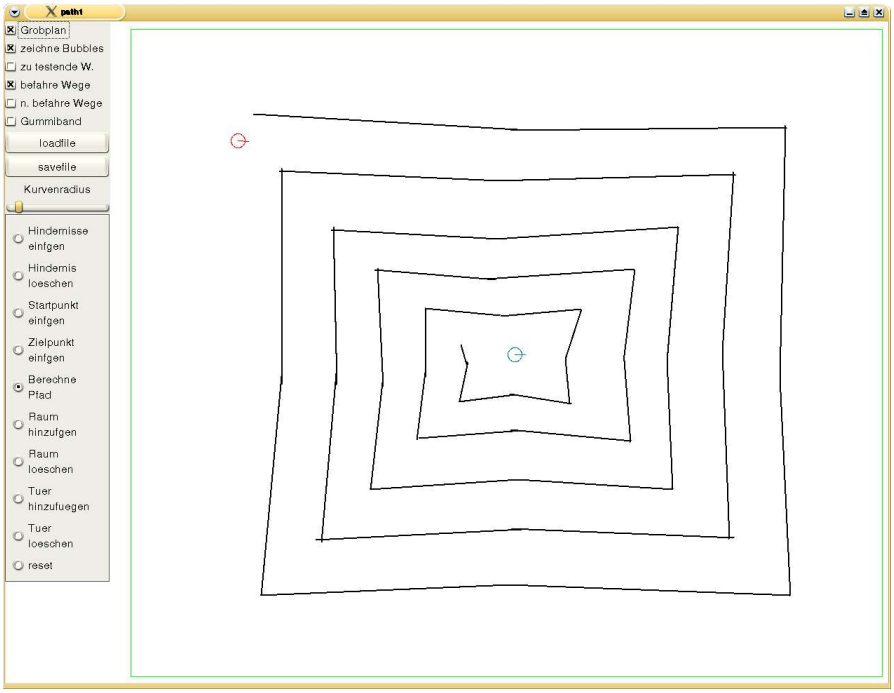


*Abbildung 38 Testszenario s20*2*

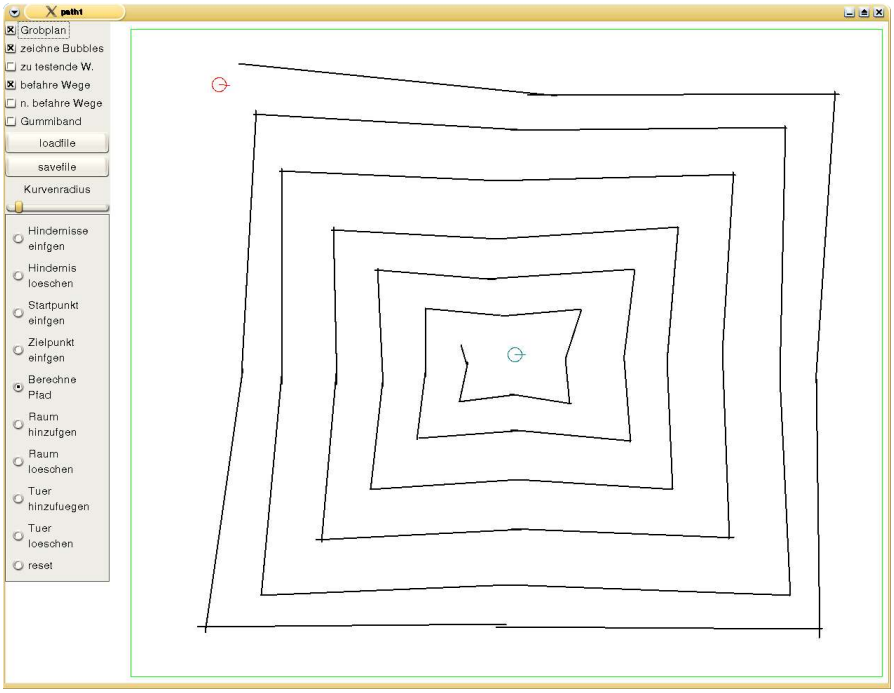
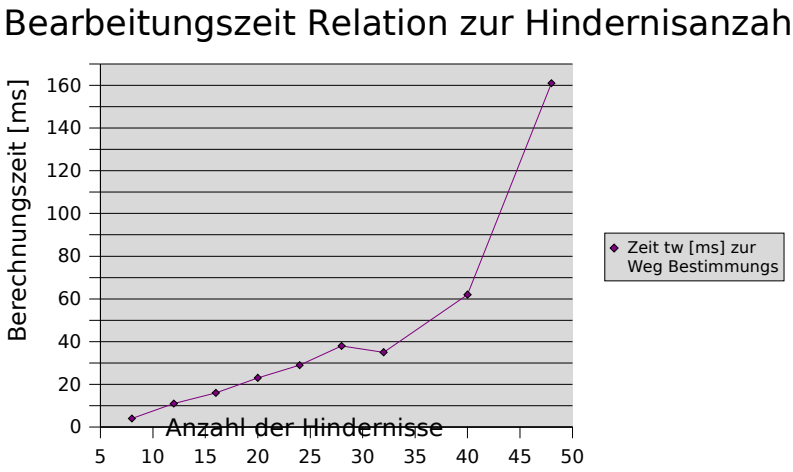


*Abbildung 39: Testszenario s24*2*

	Anzahl der Hindernisse	***Zeit t_w [ms] zur Weg Bestimmungs***	***Zeit t_B [ms] zum Erstellen der Bubbles***	***Anzahl der nicht befahrbahrer Wege n_{wnb}***	***Anzahl befahrbarer Wege n_{wb}***	***Anzahl zu testender Wege n_{wt}***	***Weglänge***	***theroretisch mögleiche Wege***
s8	8	4	22	14	17	8	805,61	120
s12	12	11	46	122	46	3	1965,44	276
s16	16	16	78	218	56	3	3657,73	496
s20	20	23	117	308	73	3	5947,26	780
s24	24	29	158	397	82	5	8767,81	1128
s28	28	38	204	519	88	0	11521,3	1540
s16*2	32	35	116	753	175	9	3585,53	2016
s20*2	40	62	183	1145	213	5	5877,99	3160
s24*2	48	161	250	2076	295	24	8711,89	4560

Tabelle 1: Vergleich der Laufzeiten in Abhänigkeit der Hindernisanzahl.



Aus der Tabelle 2 und dem Diagram ist ein eindeutig nicht linearen Zusammenhang feststellbar was ja auch unser Betrachtung für $O_1(n_o^3)$ entspricht. Da sich aber $O_1(n_o^2 * n_o)$ mit n_o für die Hindernisseerkennung und mit n_o^2 die möglichen Wegstrecken zusammensetzen, sollte sich bei Verdoppelung der Hindernisse eine Vervierfachnung der Wegstrecken ergeben. Dass dies tendenziell richtig ist, läßt sich aus dem Vergleich der Wegstrecken aus der Siumulation s12 und s24 ablesen.

s12: 177 Wegstrecken *4 =709

s24: 484 Wegstrecken

Trotz der relativ große Abweichung ist es doch ersichtlich, dass die Ordnung zwischen $O(n_o) und\, O(n_o^2)$ liegen muß.

Die Ordnung für den Befahrbarkeitstest lässt sich nicht aus den Beispielen s8-s28 erkennen da es keine aussagekräftige Zeitabweichung gibt. Deshalb haben wir drei weitere Senzarien generiert indem wir die Segmente der Spiralen aus s16 bis s24 einmal unterteilt und somit die Hinderniszahl verdoppelt haben. Der Vergleich der Werte aus s20 und s20*2 lässt nun den Schluss zu, daß der Einfluß auf die Gesamtordnung nicht sehr groß sein kann. Da sich die Wegstrecken fast vervierfachen und die Zeit sich nicht mal um den Faktor 4 erhöht, obwohl sie bei einer $O_1(n_o^3)$ annährend das Achtfache sein sollte.

Dass in den Messergebnissen von $O_1(n_o^3)$ auch der Verifikationstest $O_2(n_o^2)$ enthalten ist - auch diese Oberschranke richtig.

Dass sich Ordnung $O_3(n_o^2)$ nicht wie im worst case Fall ergibt, ist aus dem Vergleich der Laufzeit von s24 und s24*2 bei annährend gleicher Weglänge ersichtlich. Beim Vergleich fällt auf, dass sich die Laufzeit dabei mit 158ms bei S24 und 250ms s24*2 nicht einmal verdoppelt und damit einer Ordnung $> O_3(n_o)$ entsprechen würde.

9.8 Laufzeiten bei Szenarien mit unterschiedlicher Stuktur

Bis jetzt haben wir die Laufzeit in Szenarien betrachtet, wo in Folge der Aufgabenstellung das Verfahren immer möglichst schlechte Ergebnisse liefert. Nun wollen wir anhand von Beispielen zeigen, dass das Verfahren im Normalfall bessere Eergbnisse liefert. Dazu werden drei Szenarien getestet.

1) Szenario „Office2“ Abbildung 40
   Dabei handelt es sich um eine Umgebung mit vielen kleinen Hindernissen wie sie z.B. bei einer ungenauen Hinderisserkennung durch Ultraschalsensoren entstehen könnten.
2) Szenario „verteilte Hindernisse“ Abbildung 41
3) Senzario „Office5“
   Dabei handelt es sich um das Szenario „Office2“ Abbildung 42 aber mit Unterteilung in toplogische Räume.

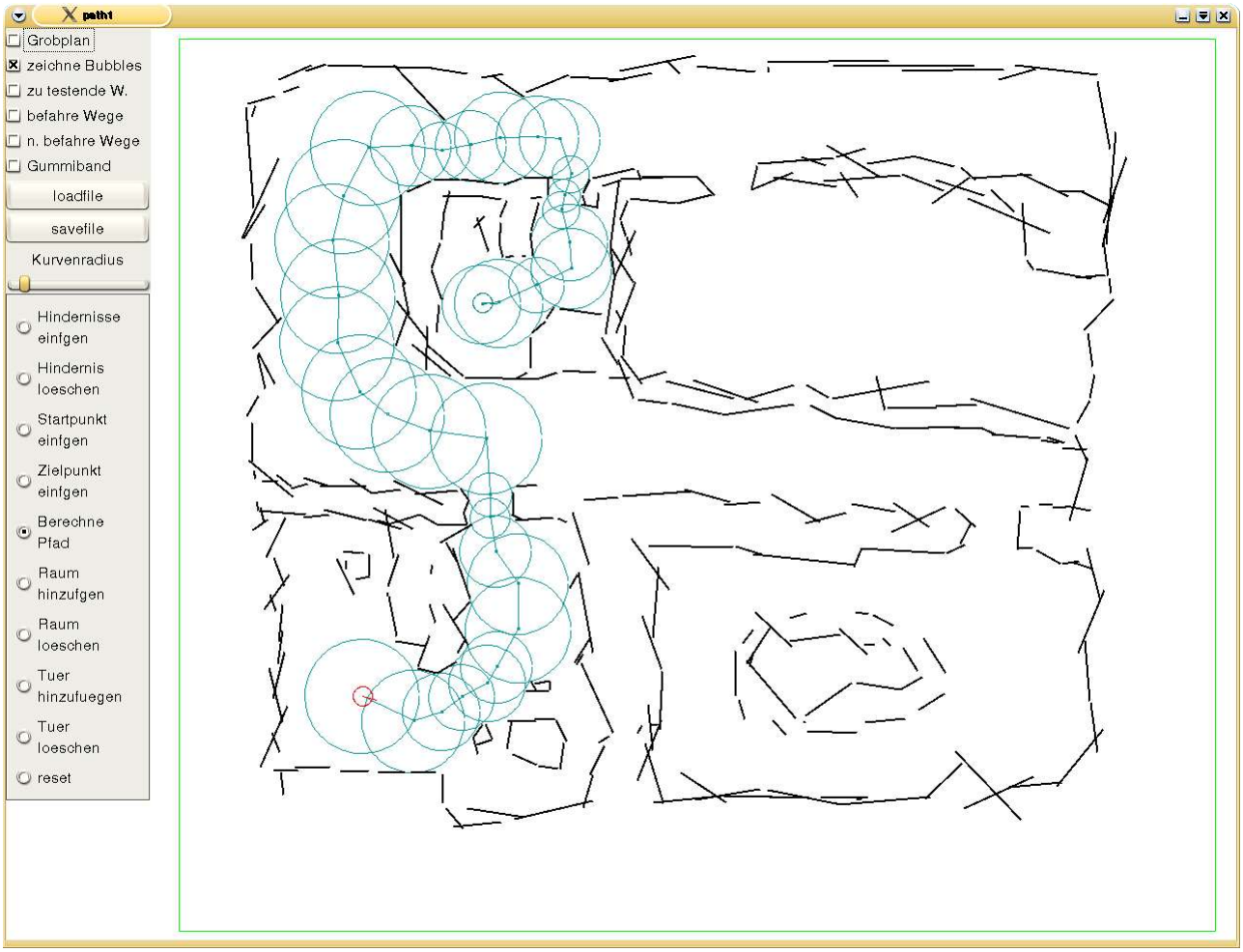


Abbildung 40: Testsenzario Office 2

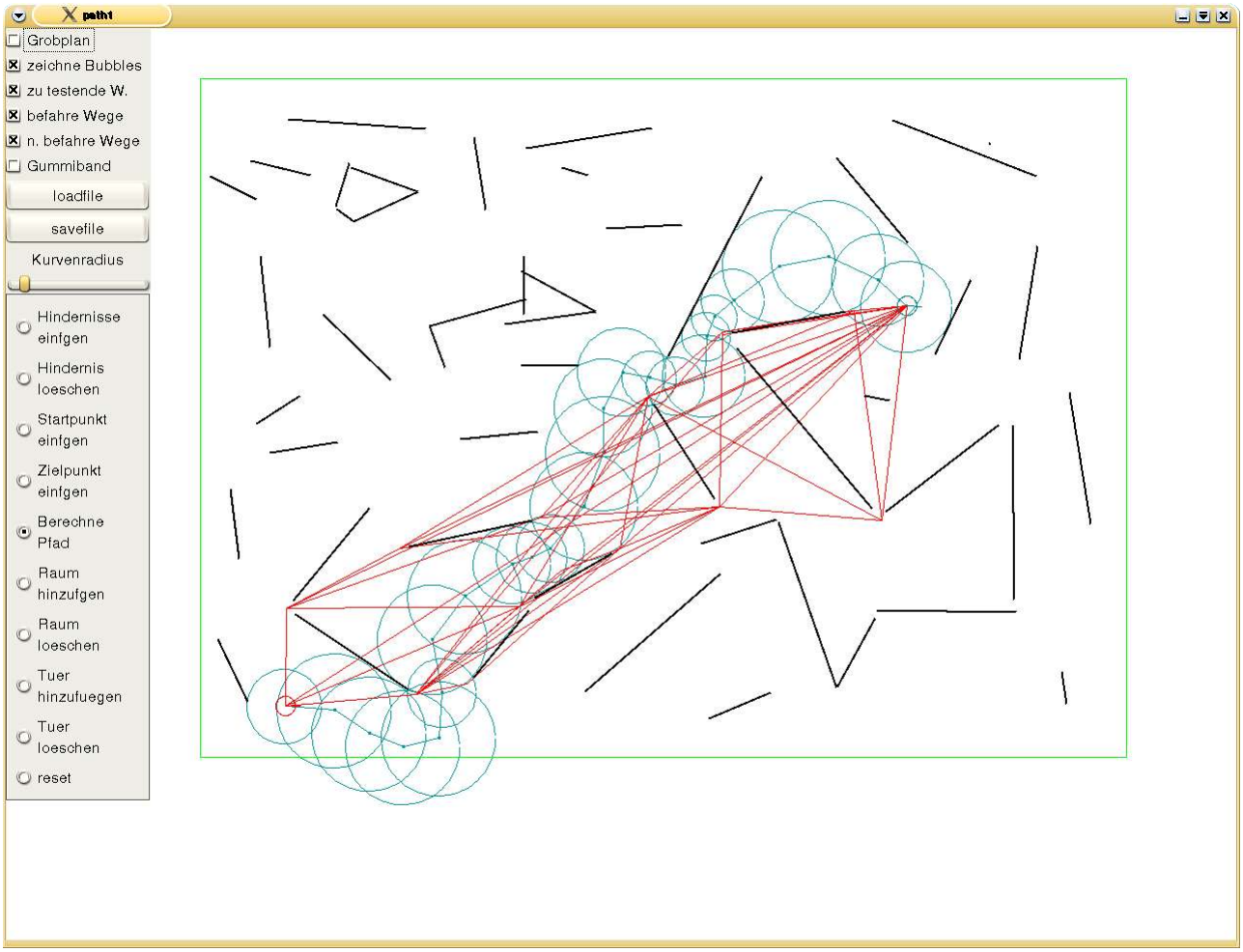


Abbildung 41: Testszenaorio "verteilte Hindernisse" mit allen berechneten Wegstrecken

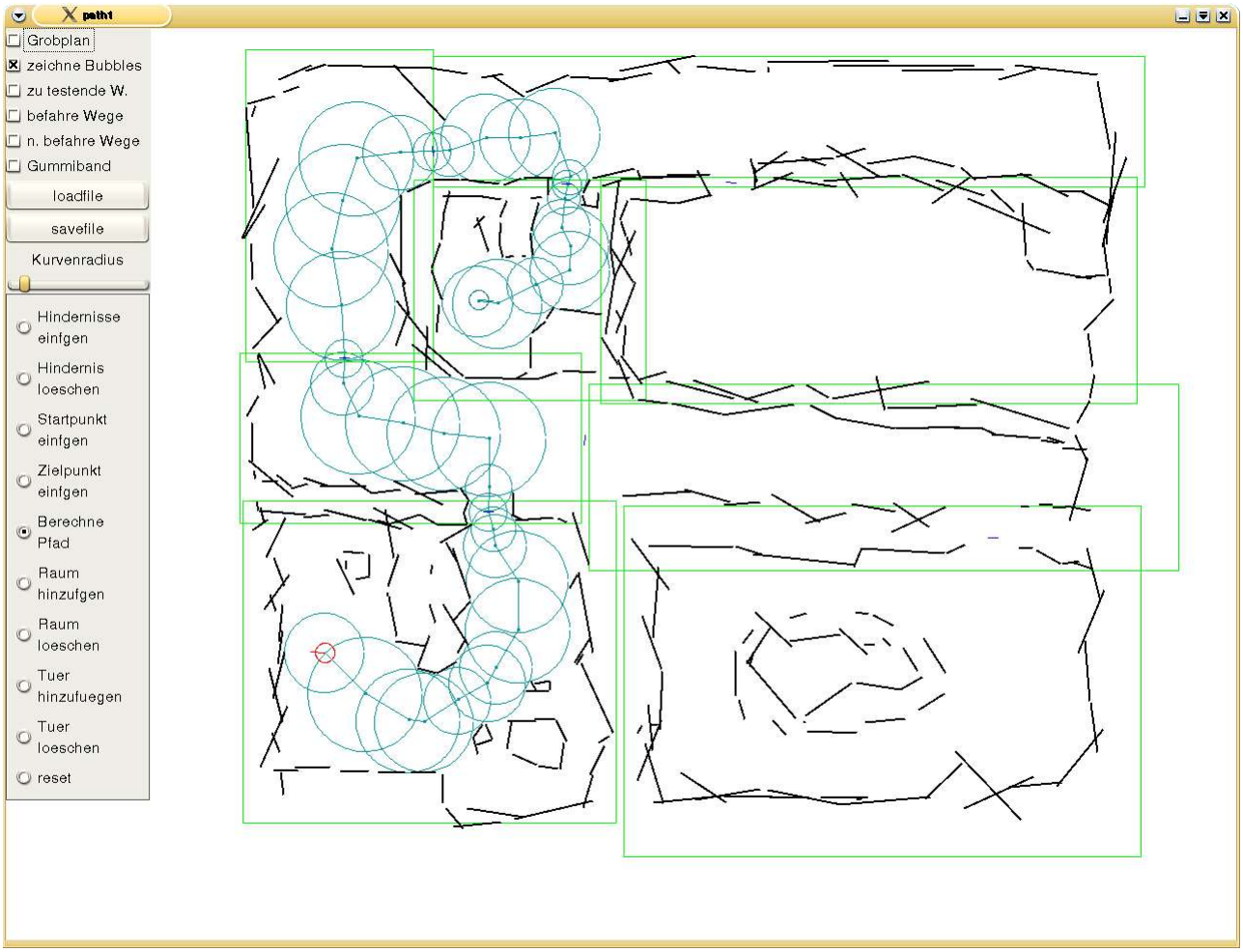


Abbildung 42: Testszenarion "Office5"

	Anzahl der Hindernisse	***Zeit t_w [ms] zur Weg Bestimmungs***	***Zeit t_B [ms] zum Erstellen der Bubbles***	***Anzahl der nicht befahrbahrer Wege n_{wnb}***	***Anzahl befahrbarer Wege n_{wb}***	***Anzahl zu testender Wege n_{wt}***	***Weglänge***	***theroretisch mögleiche Wege***
verteilte Hindernisse	49	8	45	14	24	6	1020.12	4753
Office2	333	1884	309	4230	1456	13	962.53	221445

	Anzahl der Hindernisse	***Zeit t_w [ms] zur Weg Bestimmungs***	***Zeit t_B [ms] zum Erstellen der Bubbles***	***Anzahl der nicht befahrbahrer Wege n_{wnb}***	***Anzahl befahrbarer Wege n_{wb}***	***Anzahl zu testender Wege n_{wt}***	***Weglänge***	***theroretisch mögleiche Wege***
Office5	333	65	404				1066	

Tabelle 2: Vergleich der Laufzeiten bei Szenarien mit unteschiedlichen Hindernisstrukturen

Wie aus dem Senzario „verteilte Hinderinsse“ hervorgeht, hängt die Laufzeit nicht so sehr von der Anzahl der Hindernisse ab, sondern von der Komplexität der Umgebung. Dabei hängt die Komplexität vor allem mit der Abschätzung für den günstigtsen Weg zusammen und die liegt genau dann richtig, wenn keine größen Umwege gemacht werden, wie das im Senzario „verteilte Hinderinsse“ der Fall ist. Darin liegt auch der riesige Zeitunterschied bei der Laufzeit, der zwischen den Szenarien Office2 und Office5 entsteht. Durch die Unterteilung des Szenarios in Räume werden solche Spiralen entwirrt und die geometrische Suche beschränkt sich auf die Suche in einfachen Räumen.

10 Diskussion der Probleme anhand von Beispielen in der Testumgebung

Wie bei jedem Versuch, die reale Welt in eine Korsett aus Regeln zu stecken, treten durch die Abstraktion Situationen auf, die in der realen Welt so nicht vorhanden sind. Nachfolgend soll anhand von Beispielen gezeigt werden, wo einige dieser Probleme liegen.

10.1 Der Weg führt druch eine Engstelle

Wege können bei unserem Verfahren durch Engstellen führen obwohl ein besserer Weg diese vermeiden könnte. Das Problem lässt sich mit dem Verfahren auch nicht lösen, da der Algorithmus auch keine Alternativen (Ausweichstrecken) vorschlägt, da die Wegstrecke (S,G) keine Hinderisse schneidet. Dies ist in Abbildung 43 zu sehen. Gleichzeitig zeigt das Beispiel auch, dass durch die Abstraktion des Pfades durch Bubbles ein Verlust der befahrbaren Breite auftritt. Die beiden Schnittpunkte liegen immer näher zusammen als einer der Bubbledurchmesser. Deshalb muß ein Mindestabstand zwischen den Bubbles und eine entsprechende Toleranz für die Pfadbreite vorgegeben werden.

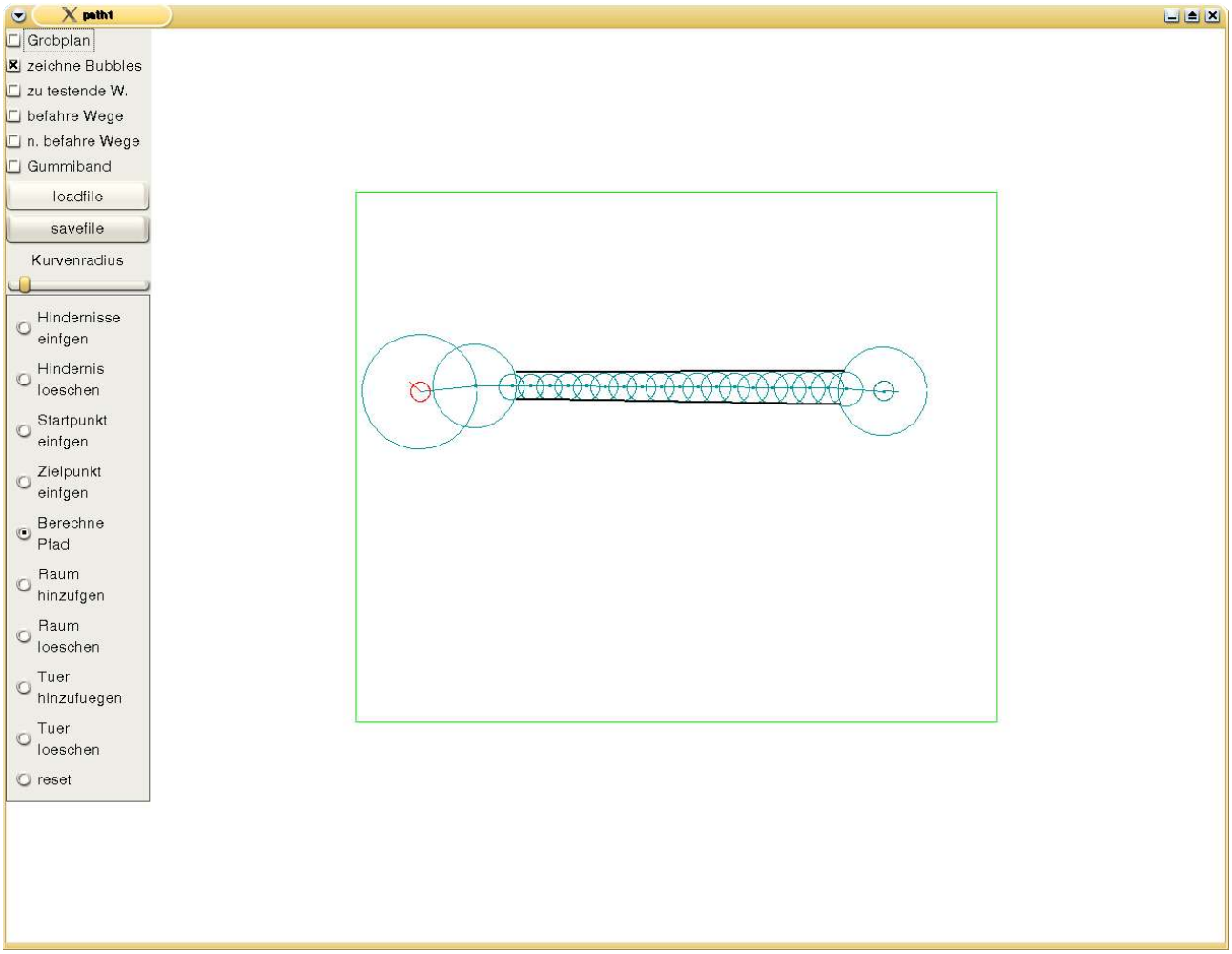


Abbildung 43: Problemfall Eingstelle

10.2 Probleme bei der Festlegung der Position der Bubbles am Start und Ziel

Da die Bubbles entlang der Wegführung verschoben werden, ergibt sich für die Start- und Zielposition eine Situation, in der die Bubble nicht verschoben werden darf. Das heißt, die Bubble muß den AMR vollständig einschliesen. Damit reduziert sich im schlimmsten Fall der Durchmesser der ersten Bubble auf den AMR Durchmesser wie es in Abbildung 44 gezeigt wird.

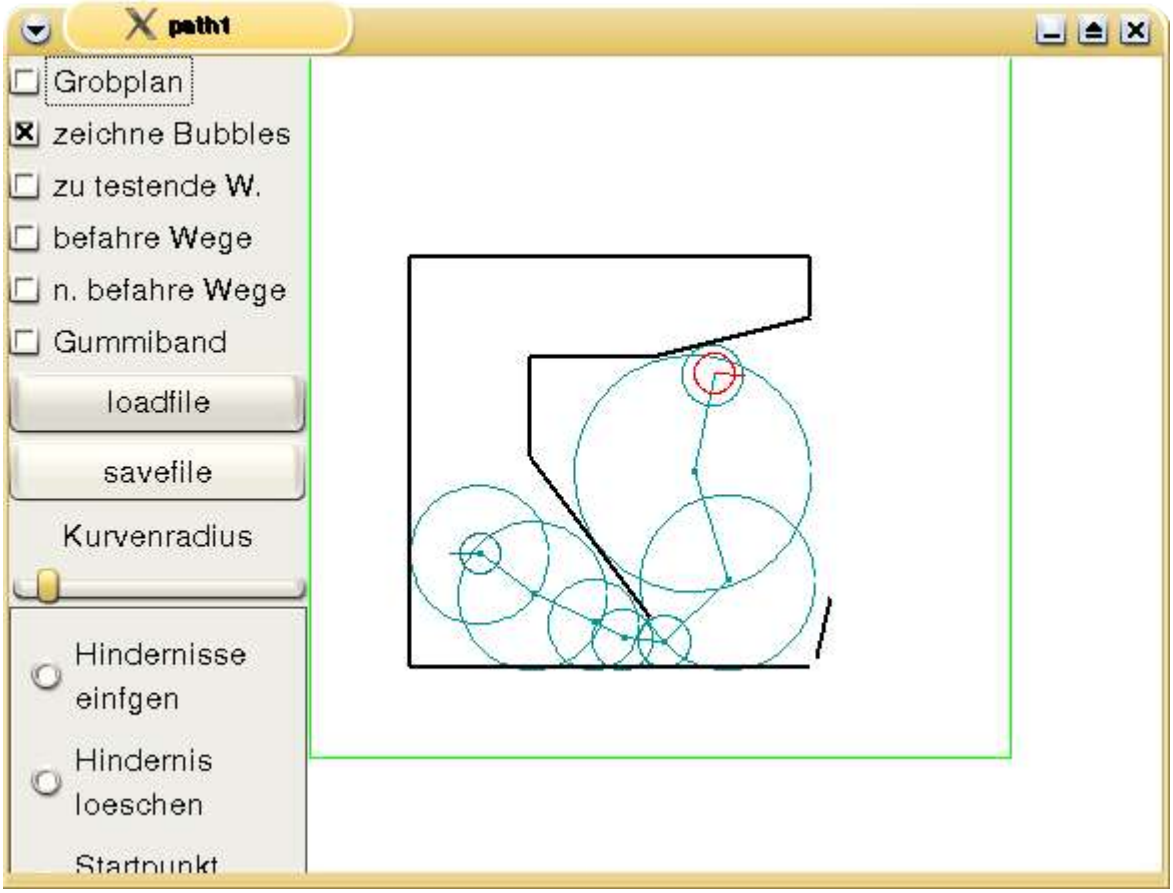


Abbildung 44: BubblePositionierungsproblem am Startl

10.3 Probleme mit dem Bubbleabstand

Da der Abstand für die Konstruktion der Bubbles entlang eines Weges in konstanten Abständen erfolgt, kann es vorkommen, dass sich die eine Bubble nach rechts bewegt und die nächste nach links. Dadurch kann es vorkommen, dass bei Engstellen der Schnittpunktabstand zweier Bubbles kleiner als die AMR-Breite wird. Dieses Problem lässt sich aber durch zusätzliche Bubbles lösen. Abbildung 45 zeigt einen solchen Fall.

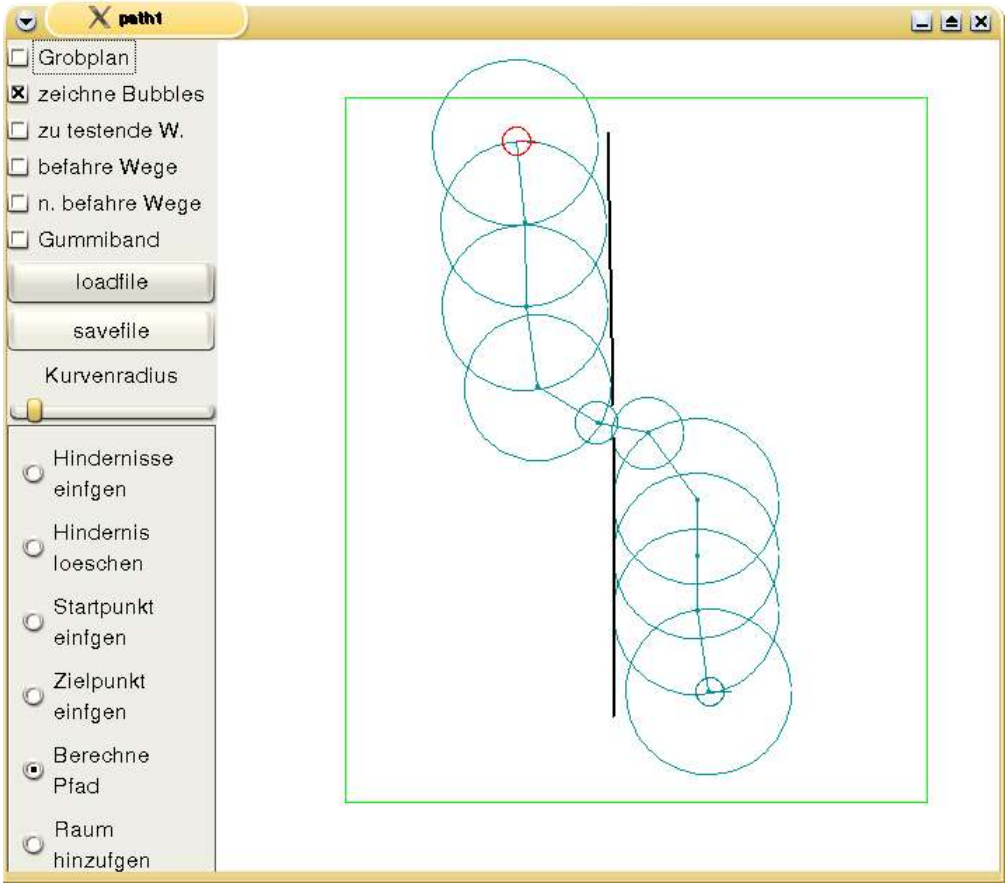


Abbildung 45: Problem mit Bubbleabständen

11 Literaturverzeichnis

[Aho83]	Alfred V. Aho, John E. Hopcroft, John E. Hopcroft, Jeffrey D. Ullman, „data Strukturs and Algorithms“,, Reading, Massachusetts, 1983
[Arkin98]	Arkin, Ronald C., Behaviour-Based Robotics, 1998, MIT Press, Cambridge, ISBN 0-262-01165-4 S.89ff & 143ff
[Crowley85]	James L. Crowley, „Navigation for an Intelligent Mobile Robot“, IEEE Journal of Robotics and Automation, Vol Ra-1, No.1, March 1985 S.31-41
[Hart68]	Peter E, Hart, Nils J. Nilsson, Bertram Raphael „A Formal Basis for heuristic Determiation of Minimum Cost Paths“ IEEE Transaction on Systems Science and Cypernetics, Vol SCC-4 No.2 July1968, S100-107
[Hoppen92]	Peter Hoppen, Autonome Mobile Roboter: Echtzeit Navigation in bekannter und unbekannter Umgebung – Mannheim; Leipzig; Wien; Zürich: BI-Wiss. Verlag, 1992 (Reihe Informatik; Band 87) ISBN 3-411-15751-8
[Laue04]	Tim Laue,“Eine Verhaltenssteuerung für autonome mobile Roboter auf der Basis von Potentialfeldern“ Diplomarbeit an der Universität Bremen,Fachbereich Mathematik und Informatik,Arbeitsgruppe von Prof. Dr. Bernd Krieg-Brückner ‚Jänner2004, http://b-smart.informatik.uni-bremen.de/public/images/d/d4/timlaue-diplom.pdf
[LoPe81]	Tomás Lozano-Pérez „Autonomic Planning of Mainpulator Transfer Movements“, IEEE Transaction an Systems, Man and Cypernetics, Vol. SMC -11, NO.10, 1981 S. 681 -698
[LoPe83]	Tomás Lozano-Pérez „Spatial Plannig: „A confiuration Space Approach“ IEEE Transactions on Computers, Vol. C-32, No.2, Feb.1983, S 108-120
[Mora80]	Hans P. Moravec, „Obstacle Avoidance and Navigation in real World by a Seeing Robot Rover“, Tecnical Rebort CMU-RI, Carnegie.Mellon Robotics Institute, pittsburgh PA, 1980
[QuKh93]	S. Quinlan, O. Khatib, "Elastic Bands: Connecting Path Planning and Control", IEEE International Conference on Robotics and Automation, Vol 2, pp.802-807, Atlanta, USA, 1993
[Stroustrup]	Bjarne Stroustrup,The C++ Programming Language, Special Edition Addision – Wesley, Boston ISBN 0-201-70073-5
[Stentz94]	Anthony Stentz, Map-Based Strategies for Robot Navigation in Unkonnown Enviroments Robotices Institute; Carnegie Mellon University; Pittsburgh, Pennsylvenia 15213 U.S.A.

[Stentz94]	Anthony Stentz,Optimal and Efficent Path Planning for Partially-Known Environments, The Robotices Institute; Carnegie Mellon University; Pittsburgh, Pennsylvenia 15213 U.S.A.
[Thrun96]	Sebastian Thrun, Arno Brücken; Integrating Grid-Based and Topological Maps for mobile Robot Navigation
[Kha86]	Khatib, O. (1986). Real-time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research S. 90-98 1986*
[Kit03]	John Kitzinger, „The Visibility Graph Among Polygonal Obstacles: a Comparison of Algorithms“, M. S., Computer Science, University of New Mexico, 2003, http://www.cs.unm.edu/~treport/tr/03-05/Kitzingerthesis.pdf

12 Anhang

- CD mit Programmquellen und Testszenarien