Projekt Robotik,
Wandverfolgung

Attila Marxer, Christian Wilkin

Projektarbeit

Februar 2005

Betreuung:

Prof. Dr. Peter Gemmar

Fachbereich Design und Informatik
Fachhochschule Trier
University of Applied Sciences

FACHHOCHSCHULE **TRIER**

Hochschule für Technik, Wirtschaft und Gestaltung
University of Applied Sciences

Autoren:	**Attila Marxer, Christian Wilkin**
Titel:	**Projekt Robotik, Wandverfolgung**
Studiengang:	**Informatik, Bachelor of Science**
Betreuung:	**Prof. Dr. Peter Gemmar**

Februar 2005

Es wird hiermit der Fachhochschule Trier (University of Applied Sciences) die Erlaubnis erteilt, die Arbeit zu nicht-kommerziellen Zwecken zu verteilen und zu kopieren.

Unterschrift des Autors

Unterschrift des Autors

An dieser Stelle möchten wir uns bei den Personen bedanken, die uns bei der Erstellung dieser Arbeit unterstützt haben. Ein besonderes Dankeschön richtet sich an Prof. Dr. rer.nat. Dipl.-Ing Peter Gemmar der uns mit seinem Fachwissen stets zur Seite gestanden hat. Herzlich bedanken möchten wir uns auch bei Dipl. Inf. Dirk Simon, für die gute Einführung in LaTeX und Anja Batz für das mehrfache Korrekturlesen.

Kurzfassung

Die vorliegende Dokumentation beschreibt die Bearbeitung des Robotikprojektes „Wandverfolgung“. Hierbei fährt der Roboter aus seiner Ursprungsposition los, bis er auf eine Wand trifft. Der Wand wird anschließend gefolgt. Der Abstand zur Wand kann durch den Benutzer beim Aufruf der Methode mitgegeben werden.

Inhaltsverzeichnis

1 Einführung

1.1 Problematik

Unsere Projektarbeit im Rahmen der Veranstaltung *Labor Robotik* im WS 04/05 beschäftigt sich mit einem Teilproblem im Zusammenhang mit der Navigation eines mobilen Roboters. Konkret geht es darum ein Verhalten zu entwickeln, das den Roboter in die Lage versetzt eine Wand zunächst zu identifizieren und ihr anschliessend zuverlässig zu folgen.

1.2 Motivation

Eines der fundamentalen Probleme bei der Entwicklung mobiler Roboter ist die Navigation in einer bestimmten Umgebung. Dazu benötigt er eine Vorstellung über das Aussehen und die Struktur seiner „Welt“ um daraus eine Strategie zu entwickeln sich kollisionsfrei darin zu bewegen. Eine Quelle der Inspiration zur Lösung dieses Problems ist das menschliche Verhalten in derselben Situation. Ist die „Welt“ statisch, so können wir uns darin anhand einer metrischen Karte orientieren, einer Piratenschatzkarte beispielsweise, die genaue Distanzen und Bewegungsanweisungen enthält.
Nun ist aber die „Welt“ generell nicht zwangsläufig starr, sondern sie besteht sowohl aus statischen als auch dynamischen Elementen, ändert sich also mit der Zeit. Wie wir selbst, besitzt also auch ein Roboter nur verlässliches Wissen über seine direkt wahrgenommene, lokale Umgebung. Bewegen wir uns beispielsweise in einer Stadt, so orientieren wir uns anhand von Merkmalen der Umgebung, sog. Landmarken, die wir evtl. mit einer Karte abgleichen um unsere Position zu bestimmen, uns also zu „lokalisieren“. Auch bei der „Navigation“ bedienen wir uns topologischer Eigenschaften. Wir bewegen uns beispielsweise in einer U-Bahn von einem Ort zum Anderen ohne deren geographische Lage zueinander, ihren geometrischen Zusammenhang, zu kennen. Auch der Roboter ist in der Lage sich mittels Sensoren verlässliches Wissen über seine lokale Umgebung zu verschaffen. Und er weiss anhand eines topologischen Weltmodells welche anderen Orte evtl. über Umwege erreichbar sind, ohne ihre Lage zueinander zu kennen. Damit er sich also von einem Ort zum Anderen bewegen kann muss er anhand von Umgebungsmerkmalen navigieren. Dabei stellen sich Aufgaben wie beispielsweise die Tür im Raum zu erkennen, hindurch zu fahren um anschliessend der Wand des Flurs zu folgen. In diesem Zusammenhang steht also die Wandverfolgung.

Sie soll es einem Roboter gemeinsam mit anderen elementaren Verhaltensweisen ermöglichen in einer unbekannten, sich ändernden Umgebung zu navigieren.

1.3 Zielsetzung

Diese Projektarbeit setzt es sich also zum Ziel ein Verhalten zur Verfolgung einer Wand zu realisieren um es dem Roboter zu ermöglichen auch ausserhalb einer Laborumgebung (mit kontrollierbaren Bedingungen und ohne äussere Einflüsse) zu navigieren. In einer unbekannten „Welt“ also, anhand einer topologischen Karte. Wie in Kapitel 3.2 beschrieben, soll das Verhalten so stabil und verlässlich wie möglich mit allgemeinen, konvexen als auch konkaven Wandverläufen umgehen können.

2 Hard- und Softwareumgebung

[SiKl03] Dieses Kapitel beschreibt die Soft- und Hardwareumgebung in die das Verhalten zur Wandverfolgung eingebettet ist. Es wird die Aria/Saphira Entwicklungs- und Simulationsumgebung vorgestellt, ihre grundlegende Funktionalität und Arbeitsweise beschrieben. Auch wird auf die eingesetzte Hardware eingegangen, den PIONEER 2DX für den das Verhalten entwickelt wird.

2.1 Pioneer 2DX8

Verwendet wurde ein Roboter **Pioneer 2DX8** von der Firma *Activmedia Robotics*. Er ist standardmäßig mit einem Differentialantrieb ausgestattet, welcher 2 Räder und ein *Castor-Wheel* antreibt. Durch sogenannte *Shaft-Encoders* kann er Selbstlokalistation durchführen, was auch als **Advanced Dead-Reckoning** bezeichnet wird. Insgesamt ist er mit 16 Sonarsensoren ausgestattet. Das sind 8 Stück an der Vorder- und an der Rückseite, wovon 2 jeweils an den Seiten angebracht sind. Die 6 anderen stehen im 20 Grad Winkel zueinander. Die Reichweite der Sonarsensoren erstreckt sich von 10 cm bis maximal 4 Meter.
Der Roboter ist mit einem *ONBOARD-PC* ausgestattet, der einen 850Mhz handelsüblichen Pentium III enthält. Das Board ist ein Formfaktor EBX-Board und hat alle Eingabemöglichkeiten, die ein normaler PC auch hat. Mit der W-LAN-Schnittstelle kann der Zugang zum Internet bzw. zu einem LAN hergestellt werden kann.

2.2 Der Differentialantrieb

Der Differentialantrieb des Pioneer-Roboters erlaubt, gemeinsam mit dem dritten Rad, dem Castor-Rad, eine holonomische Bewegung des Roboters. Er muss also lediglich θ, seine aktuelle Richtung ändern durch Drehung um die eigene Achse, um aus einer gegebenen Position (x,y,θ) eine andere Position anzufahren.

Es handelt sich bei einem Differentialantrieb um zwei Räder, die an derselben Achse montiert, unterschiedlich angesteuert werden, im konkreten Fall sogar von zwei verschiedenen Motoren. (Siehe hierzu [?]) Dieser Antrieb ermöglicht eine relativ einfache mathematische Modellierung. Bedingt durch diese Konstruktion, kann der Roboter sich nur auf Kreisbahnen bewegen, deren Mittelpunkt (ICC)

auf der Verlängerung der Achse des AMR liegen Eine Geradeausfahrt wäre dann eine Kreisbahn mit unendlichem Radius mit identischen Geschwindigkeiten der beiden Räder v_{rechts} und v_{links}. Letztere bestimmen die entstehende Trajektorie, sind aber darin nicht unabhängig voneinander. Der AMR bewegt sich um einen ICC entlang eines Pfades mit einer Winkelgeschwindigkeit ω (Darstellung nach Dudek und Jenkin [?]).

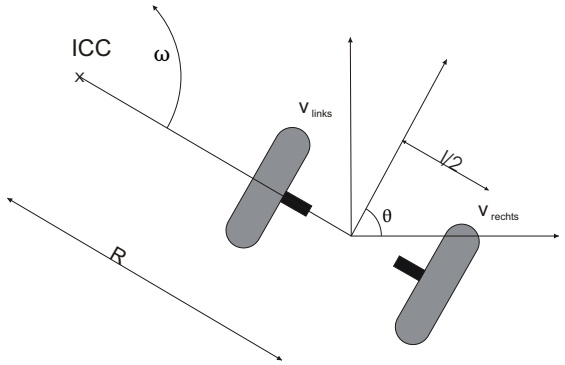


Abbildung 2.1: Kinematik des Differentialantriebs

So ergeben sich die Geschwindigkeiten für die Räder als (nach: [Gem05])

$$v_{rechts} = \omega(R + l/2) \tag{2.1}$$

$$v_{links} = \omega(R - l/2) \tag{2.2}$$

mit l als Achsenbreite, und R als Distanz zwischen Achsmittelpunkt des AMR und seinem ICC. So ergibt sich R als

$$R = l/2 * (v_{rechts} + v_{links})/(v_{rechts} - v_{links}) \tag{2.3}$$

und ω als

$$\omega = (v_{rechts} - v_{links})/l \tag{2.4}$$

Der ICC des AMR lässt sich aus bekanntem Radius (Abstand zwischen AMR und ICC) und Position (x,y,θ) bestimmen als

$$ICC = [x - R\sin(\theta), y + R\cos(\theta)] \tag{2.5}$$

Für die mittlere Winkelgeschwindigkeit ω während eines bestimmten Zeitabschnittes δ t errechnet sich die neue Position des AMR durch:

$$\begin{bmatrix} x \\ y \\ \phi \end{bmatrix} = \begin{bmatrix} \cos(\omega\,\delta\,t) & -\sin(\omega\,\delta\,t) & 0 \\ \cos(\omega\,\delta\,t) & -\sin(\omega\,\delta\,t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x - ICCx \\ y - ICCy \\ \phi \end{bmatrix} + \begin{bmatrix} ICCx \\ ICCy \\ \omega\,\delta\,t \end{bmatrix} \quad (2.6)$$

$$= \begin{bmatrix} \cos(\omega\,\delta\,t)\,(x - ICCx) - \sin(\omega\,\delta\,t)\,(y - ICCy) + ICCx \\ \cos(\omega\,\delta\,t)\,(x - ICCx) - \sin(\omega\,\delta\,t)\,(y - ICCy) + ICCy \\ \phi + \omega\,\delta\,t \end{bmatrix} \quad (2.7)$$

Die Drehung mit der Rotationsmatrix um die vertikale Raumachse z erfolgt hier zunächst normiert um den Ursprung. Danach wird mit dem Abstand zum ICC multipliziert und so auf die konkreten Längeverhältnisse skaliert. Zuletzt erfolgt die Translation des ICC an seinen tatsächlichen Ort im Koordinatensystem durch Addition.

2.3 Die Entwicklungs- und Simulationsumgebung Saphira

„Saphira“ wurde für den Roboter **Flakey** entworfen und über 10 Jahre hin verwendet. Dieser Roboter ist ähnlich ausgestattet, wie der hier verwendete **Pioneer 2-DX**. Er hat ebenfalls einen Differentialantrieb, der zwei Räder unabhängig voneinander ansteuert. Außerdem besitzt er 12 Sonarsensoren und eine Videokamera (beim Pioneer 16). Hinzu kommt noch ein Laserscanner, der bei dem hier verwendeten Roboter nicht dabei ist.
Schon **Flakey** verwendete das *Fuzzy-Behaviour-System*, in welchem verschiedene Behaviours in einem synchronen Zyklus parallel arbeiten und als eine *Absicht* (*desired action*) mit einer bestimmten Stärke verbunden (*strength* auf C++-Ebene und *priority* auf Colbert-Ebene), ausgeführt werden. Erst dadurch ist es möglich einen gewissen Grad an Autonomie zu erreichen. So ist es zum Beispiel möglich, dass ein Roboter zum einen ein bestimmtes Ziel mit einer bestimmten Stärke verfolgt und andererseits autonom auf Hindernisse reagiert. Außerdem wurde durch **Flakey** bereits der *LPS=Local Perceptual Space* definiert, in welchem die Sensor-Informationen ausgewertet werden. Die Reaktion erfolgt, wenn bestimmte Voraussetzungen im *LPS* erfüllt sind. Unter Voraussetzung kann zum

Beispiel eine Wand bezeichnet werden, die als Objekt im *LPS* erkannt wird. Da dies auf *Fuzzy-Behaviours* beruht, gibt es keine Reaktion, die sich völlig durchsetzt, sondern es kommt zu verschiedenen Reaktionen, die überlagert zu einer komplexen Reaktion führen, die aus den Teilbehaviours abgeleitet wird.

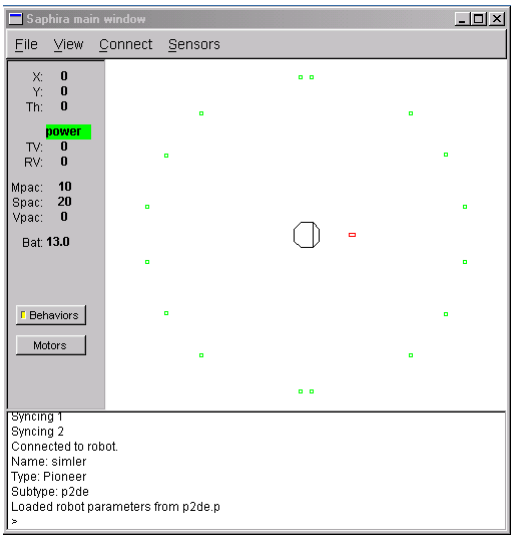


Abbildung 2.2: Saphira Screenshot

Ab Version 8 wurde „Saphira“ gesplittet. Alle sogenannten *Low-Level-Routinen* wurden nun in „Aria“ zusammengefasst und alles andere in „Saphira“ integriert. „Saphira“ ist nicht unabhängig von „Aria“, sondern baut darauf auf. Durch eine graphische Benutzeroberfläche können in „Saphira“ alle Behaviours gesteuert, die komplette Sensorik der Sonarsensoren repräsentiert,- und die Welt dargestellt werden.
Außerdem ist es möglich, auch ohne den realen Roboter zu arbeiten. Statt dessen kann eine Simulation als Client verwendet werden, die alle Eigenschaften des Roboters simuliert. Dabei wird vor allem auch die Schwäche der Selbstlokalisation berücksichtigt. Dies wird durch eine Simulation erreicht, die sogar das Radrutschen mit einbezieht.
Es gibt zwei verschiedene Kontrollmodi, die in „Saphira“ angewandt werden können. Erstens kann das o.a. *Fuzzy-Behaviour-System*-, und zweitens ein sogenanntes *Direct-Action-Command-System* verwendet werden. Diese beiden Kontrollmodi schließen sich gegenseitig aus. Letzteres steuert den Roboter durch simple Bewegungskommandos, die sich nicht überlagern können.

2.4 Die Programmierschnittstelle Aria

„Aria“ (das Akronym steht für „ActivMedia Robotics Interface for Applications“) stellt die grundlegende Schnittstelle zwischen Roboter und Programmiersprache dar. Hier ist u.a. auch die Systemarchitektur implementiert. Auf der Basis von „Aria“ ist es möglich, auch unabhängig von „Saphira“ Applikationen zu entwickeln. Zu betonen ist, dass auf der Ebene von „Aria“ die Systemarchitektur verwendet werden kann, um neue Steuerungsmöglichkeiten zu entwerfen. Auch „Saphira“ verwendet diese Möglichkeit, um das Fuzzy-Behavioursystem zu implementieren. Interessant dabei ist, dass sich zum Beispiel nicht mehr um die komplizierte Kommunikation zwischen Roboter und Client-Anwendung gekümmert werden muss. Letztere bezeichnet die Applikation, welche sich mit dem Roboter verbindet.
Eine „Aria“-Anwendung hat exklusiven Zugriff, wenn einmal die Verbindung zur seriellen Schnittstelle aufgebaut wurde. Es ist also nicht möglich, dass sich zwei Applikationen den Roboter „teilen“.

2.5 Colbert

„Colbert“ ist eine C-ähnliche Programmiersprache, die in „Saphira“ integriert ist. Mit dieser Sprache ist es möglich, simple Schleifen, Entscheidungsabfragen und *GOTO-Befehle* zu formulieren. Sie ist vor allem dafür geeignet, den Roboter durch das oben erwähnte *Direct-Action-Command-System* zu bedienen. Der andere komplizierte Steuerungsmodus wird hier nicht unterstützt. Die wichtigste Aufgabe, die „Colbert“ hat, ist die Funktion einer Skriptsprache. Es ist möglich, vorher definierte Behaviours mit einer bestimmten Priorität und in einer bestimmten Art und Weise zu starten. Sie müssen allerdings vorher in einer DLL erstellt worden sein, die dynamisch hinzu geladen werden kann. „Colbert“ ermöglicht also auch in einer einfachen Art und Weise den Zugriff auf vorher definierte Klassen. Dabei können auch globale Funktionen in der DLL aufgerufen werden, die den Ablauf der Behaviours beeinflussen. Die Sprache könnte auch als Kontrollsprache des „Saphira“-Laufzeitsystems angesehen werden.
Auch in „Colbert“ können Behaviours definiert werden. Diese können als sogenannte *act-Methoden*, auch **Activities** genannt, in „Colbert“ integriert- und den schon vorhandenen Zyklen hinzugefügt werden).

2.6 Das Micro-Tasking Betriebssystem

Das „Micro-Tasking OS“ beschreibt die Systemarchitektur des Gesamtsystems. Ein Task kann durch einen endlichen Automaten beschrieben werden, der einen bestimmten Zyklus hat. Voreingestellt ist die Dauer von 100 Millisekunden. Das heißt, dass jede $\frac{1}{10}$ Sekunde ein Schritt in jedem aktiviertem Behaviour ausgeführt wird. Das bedeutet aber darüber hinaus auch noch, dass alle anderen Routinen synchronisiert ablaufen. Hier wären auch die Paket-Kommunikation und der Status-Reflektor zu nennen, welcher den augenblicklichen Zustand des Roboters auf dem Client abbildet. Durch diese Zyklen können auch Warteschleifen integriert werden, die das Gesamtsystem nicht blockieren.

3 Aufgabenstellung

3.1 Randbedingungen

Dieses Kapitel dient dazu einige Gedanken festzuhalten bezüglich der Rahmenbedingungen und Voraussetzungen in die das Verhalten zur Wandverfolgung eingebettet ist und kann als Einführung zu Kapitel 3.2 und verstanden werden welches Anforderungen für das Verhalten definiert.

3.1.1 Interaktion mit anderen Behaviours und Zeitlimit

In der Regel werden benutzerdefinierte Verhaltensweisen wie die Wandverfolgung als sog. synchrone Client Prozesse von Saphira/Aria implementiert. Diese Verhaltensweisen werden jeden Zyklus (100ms) nach Priorität absteigend sequenziell ausgewertet und beeinflussen je nach Stärke das Gesamtverhalten in unterschiedlichem Masse. Vor dem Hintergrund dieser Gleichzeitigkeit bzw. Überlagerung von Verhaltensweisen ist es notwendig sich darüber Gedanken zu machen in wie fern die Wandverfolgung einerseits von anderen Behaviours beeinflusst wird und andererseits wiederum auf diese Einfluss nimmt. Eine wesentliche Rolle spielt bei dieser Überlegung die zeitliche Einschränkung.
Stärke: Allen Behaviours ist ein bestimmtes Stärkepotenzial (1.0 für Ausrichtung und 1.0 für Geschwindigkeit) gemein. Durch den Verbrauch eines gewissen Teils dieser Ressourcen durch einzelne Verhalten wird festgelegt wie stark diese auf die beiden Bewegungsmodi Ausrichtung und Geschwindigkeit Einfluss nehmen:

- Verhalten höherer Priorität können nicht beeinflusst werden, bzw. nur in dem Masse wie sie es selbst (über ihre Stärke) zulassen.
- Verhalten geringerer Priorität auf der anderen Seite können beeinflusst werden. Verändert das Verhalten Ausrichtung und/oder Geschwindigkeit mit einer Stärke < 1.0 so lässt es Behaviours geringerer Priorität die Möglichkeit ebenfalls noch in das Gesamtverhalten einzugehen. Werden hingegen alle Ressourcen aufgezehrt (Stärke 1.0) so „verhungern“ schwächere Behaviours.

Zeitintervall: Die Auswertung aller synchronen Client Behaviours erfolgt in einem Zeitfenster von 100ms. Im Sinne einer „fair policy“ sollte also die Wandverfolgung nur einen Teil dieser Rechenzeit für sich beanspruchen. Zeitüberschreitungen werden vom Micro-Tasking Betriebssystem behandelt. Saphira allerdings verlangt, dass jedes Verhalten „bequem“ innerhalb dieses Zeitintervalls ausgeführt werden kann. Komplexere Verhaltensweisen wie SfGrad oder die Markov Lokalisation,

die auf recht umfangreichen Berechnungen aufbauen, sollten daher als sog. asynchrone Routinen implementiert werden die als eigenständige Threads unabhängig vom Zyklenintervall ablaufen.

3.1.2 Beurteilung der Wahrnehmung

Der AMR nimmt die Welt und damit auch die Wand ausschliesslich punktuell in Form sog. Sonarreadings wahr und trifft eine Annahme über das Aussehen der lokalen Umgebung in Form eines Modells. Aufgrund dieser Approximation ergeben sich zwangsläufig Abweichungen von der realen Situation, bedingt auch durch die Klassifikation der Readings. Dabei wird jedes einzelne von ihnen nach seiner Korrektheit und Relevanz für das Problem beurteilt.

- *korrektes Reading*. Ein Reading das Träger korrekter Information über die Umwelt ist und dazu beiträgt die Genauigkeit des Modells zu verbessern.
- *Phantomreading*. Ein Reading das ein real nicht existierendes Hindernis andeutet und dadurch die Genauigkeit des Modells mindert.
- *Ausreisser*. Ein Reading welches ein existierendes Hindernis verfälscht wiedergibt.
- *Irrelevantes Reading*. Ein Reading das für das gegebene Problem nur bedingt relevant oder gar irrelevant ist.

Die Klassifikation von Readings ist ein schwieriges Problem, da sie implizit ein bestimmtes Wissen über die Struktur der Umwelt voraussetzt. Sie basiert also wiederum auf Annahmen und entsprechenden Wahrscheinlichkeitsmodellen. Eine solche mögliche Annahme wäre es beispielsweise davon auszugehen, dass sich Objekte als Polygonzüge darstellen lassen, also stückweise linear sind. Treten nun aber nichtlineare Kantenzüge auf, so führt dies zwangsläufig zu einer fehlerhaften Beurteilung der Sonardaten.

3.1.3 Eingeschränkte Wahrnehmung (blind spots)

Obwohl der PIONEER 2DX aufgrund der Anordnung seiner Sensoren einen Sichtbereich von 360° abdeckt zeigt die Praxis, dass bestimmte Winkelbereiche kaum verlässliche Informationen über die Umwelt liefern. In Bereichen um $\pm 45°$ treten beispielsweise tote Winkel auf, sog. blind spots, Bereiche also die meist keine Readings liefern auch wenn dort Hindernisse existieren. Dies erschwert es dem Verhalten eine Vorstellung seiner Umgebung zu entwickeln.

3.2 Anforderungen

Diese Kapitel widmet sich der Beschreibung von Anforderungen an das Verhalten zur Wandverfolgung. Es dient dazu Schwerpunkte bzw. Richtlinien zu beschreiben an denen sich eine mögliche Lösung orientieren bzw. messen soll. Es ist klar, dass nicht alle hier formulierten Forderungen vollständig erfüllt werden können zumal sie sich teilweise widersprechen. Dennoch sollen sie als Leitfaden dienen und eine praktikable, praxistaugliche Lösung ermöglichen.

3.2.1 Planung

Bei der Verfolgung einer Wand handelt es sich prinzipiell um ein lokales Problem zu dessen Lösung keine vollständige Kenntnis der Welt, insbesondere kein Weltmodell benötigt wird. Zudem wird eine Planung durch die in Kapitel 3.1.3 erwähnten toten Winkel erschwert. Daher soll es als effizientes, reaktives Verhalten ohne Planung implementiert werden das dafür aber schnell und direkt auf Änderungen in der lokalen Umgebung eingeht.

3.2.2 Dominanz

In der Regel ist es wünschenswert, dass in einer bestimmten Situation auch ein bestimmtes Verhalten dominant ist um überhaupt eine Aussage über das Gesamtverhalten machen zu können. Daher soll die Wandverfolgung, wenn aktiv, alle noch verfügbaren Ressourcen für sich vereinnahmen, also beide Bewegungsmodi (Geschwindigkeit und Ausrichtung) mit der Stärke 1.0 kontrollieren. Der Benutzer soll sich darauf verlassen können, dass sich das Verhalten, wenn aktiv, auch durchsetzt.

3.2.3 Effizienz

Das Verhalten ist dazu gedacht zusammen mit anderen Grundverhaltensweisen zu einem komplexen Gesamtverhalten beizutragen. Aufgrund des in Kapitel 3.1.1 erwähnten, begrenzten zeitlichen Rahmens, soll daher das Verhalten so kompliziert wie nötig, aber so einfach wie möglich umgesetzt werden. Das Verhalten soll also den Berechnungsaufwand minimieren da nicht absehbar ist wieviele andere Verhaltensweisen im selben Zyklus ausgeführt werden. Dazu sollen auch Symmetrieeigenschaften des Problems ausgenutzt werden.

3.2.4 Robustheit

Die Wandverfolgung soll möglichst adaptiv und vielseitig einsetzbar sein. Sie soll mit jeglicher Art von Wandverlauf umgehen können und kaum auf Wissen über die Umwelt zurückgreifen. In der Praxis trifft man jedoch meist auf bestimmte vorherrschende Wandverläufe die sich in Form von Polygonzügen zeigen. Es soll daher als Mindestanforderung an eine Lösung gelten, dass sie mit den folgenden besonderen Wandverläufen umzugehen weiss.

- gerader Wandverlauf.

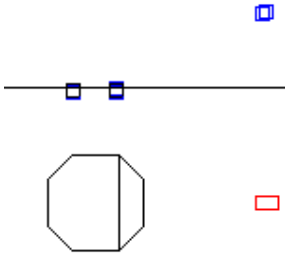


Abbildung 3.1: gerader Wandverlauf

- allgemeiner konvexer Wandverlauf.

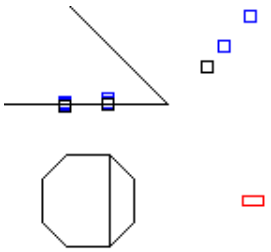


Abbildung 3.2: allgemeiner konvexer Wandverlauf

- spezieller konvexer Wandverlauf (Richtungsänderung um π).

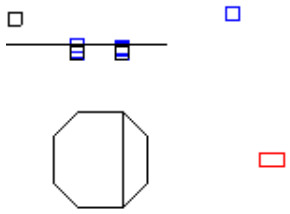


Abbildung 3.3: spezieller konvexer Wandverlauf um π

- allgemeiner konkaver Wandverlauf.

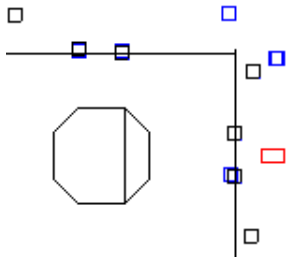


Abbildung 3.4: allgemeiner konkaver Wandverlauf

3.2.5 Stabilität

Stabilität steht in engem Zusammenhang und auch im Widerspruch mit der Forderung nach einem möglichst allgemeinen, robusten Verfahren. Wie in Kapitel 3.1.2 angesprochen ist die Klassifikation von Readings ein wesentlicher Faktor für die Stabilität des Verfahrens. Je mehr über die Struktur der Umwelt bekannt ist, umso besser können Readings klassifiziert und das Verhalten darauf angepasst werden. Eine Lösung soll als stabil betrachtet werden, wenn folgendes gilt:

- Der Roboter verliert zu keinem Zeitpunkt den Kontakt zur Wand. Dazu ist es nötig schnell und direkt auf Richtungsänderungen zu reagieren. Änderungen machen sich also unmittelbar im Verhalten bemerkbar.

- Der Roboter zeigt ein ruhiges Fahrverhalten. Geringe Änderungen im Wandverlauf zeigen sich also nicht in der Bewegung und ein ständiges Korrigieren der Ausrichtung wird vermieden.

Da sich diese beiden Kriterien grundsätzlich widersprechen, muss ein optimales Gleichgewicht zwischen ihnen gefunden werden.

3.3 Annahmen und Einschränkungen

Im Folgenden werden einige Annahmen und Einschränkungen definiert die für das Problems gelten sollen.

3.3.1 Struktur der Welt

Als Einschränkung für die Forderung in Kapitel 3.2.4 nach einem robusten Verfahren gehen wir davon aus, dass sich die Welt und die Objekte darin gut stückweise linear approximieren lassen.

3.3.2 Verantwortlichkeit

Im Grunde genommen müsste eine Wandverfolgung auch über weitreichende Fähigkeiten zur Kollisionsvermeidung, zur Erkennung von Sackgassen und seitlichen Durchgängen verfügen. Um aber ein effizientes, elementares Verhalten zu implementieren und weil diese Aufgaben nicht primärer Bestandteil einer Wandverfolgung sind, soll ihnen eine Lösung nur insofern Beachtung schenken, als dass es für die Verfolgung einer Wand unbedingt erforderlich ist.

3.4 Lösungsansätze

Bei der Wandverfolgung handelt es sich um ein symmetrisches Problem im lokalen Bezugssystem des Roboters. Das Verhalten wird aktiv sobald erstmals ein Sonarreading erfasst wurde. Anschliessend gilt als Invariante, dass sich die Wand, relativ zum Roboter, immer auf derselben Seite befindet. Eine Verletzung dieser Einschränkung würde bedeuten dass der AMR entweder eine andere Wand

verfolgt oder dieselbe in entgegengesetzter Richtung - Beides beschreibt ein unerwünschtes Fehlverhalten. Aus praktischen Überlegungen hält das Wandverfolgungs Behaviour diese Information in einer Variablen fest die die drei Werte 0, 1 oder -1 annehmen kann, je nachdem ob das Verhalten inaktiv ist oder sich die Wand links bzw. rechts bezüglich des Roboters befindet. Dadurch wird der Symmetrie Rechnung getragen indem dieser Wert als Faktor in Winkelberechnungen Verwendung findet und das Verhalten als Ganzes dadurch gespiegelt wird.

3.4.1 Lösungsansatz: Zustandautomat

Ein erster Lösungsansatz sah vor einen Zustand z als binäre Zahl zu kodieren: Jedem Sensorbereich wird dabei einen Index i aus $[0..n]$ zugeordnet. Anschliessend wird jeder Sensorbereich i auf Readings untersucht und falls vorhanden wird zum Zustand z der Wert 2^i addiert. Dadurch wird das i-te Bit in z gesetzt. Anschliessend wird für jeden Zustand bzw. bestimmte Mengen von Zuständen ein Verhalten festgelegt, beispielsweise für alle Zustände die an einer Position k ein Reading besitzen.

Stärken

- Einfache Implementierung durch die klare Trennung der einzelnen Situationen. Jeder Wandverlauf kann einzeln für sich betrachtet und gelöst werden. Es muss also kein Algorithmus gefunden werden der sich generisch allen Situationen anpasst.

Schwächen

- Die Zahl der möglichen Zustände steigt exponentiell mit der Anzahl der Sensorbereiche.
- Jeder Zustand, auch irrelevante, müssen spezifiziert werden. D.h. der Zustandsautomat muss vollständig beschrieben sein. Daraus erwächst die Gefahr einzelne Zustände zu übersehen.
- Ein solcher Zustandsautomat ist nur ein sehr ungenaues Modell für den Verlauf der Wand, da lediglich nach besetzten und unbesetzten Sensorbereichen unterschieden wird und es keine Information über Abstand und Ausrichtung der Wand enthält. Durch die geringe Zahl der Readings die in die Betrachtung einbezogen werden ist das Verhalten anfälliger gegenüber Störungen. Vor allem bei extremen Wandverläufen wird das Verhalten recht unzuverlässig (s. 3.2.4) und das Fahrverhalten sehr unruhig (s. 3.2.5).

3.4.2 Lösungsansatz: Leuchtturm

Die Idee dieses Ansatzes ist es, einen Sensorbereich analog zu einem Leuchtturmfeuer um den Ursprung des lokalen Koordinatensystems zu drehen. Dabei wird nach dem ersten Winkel α gesucht, für den der Sensorbereich kein Reading enthält. Anschliessend richtet sich der Roboter nach α aus. Um aber sicher zu stellen dass sich kein Reading im Pfad des Roboters befindet, ist es nicht möglich diesen schwenkbaren Sensorbereich durch einen Öffnungswinkel zu beschreiben und über *getClosestPolar* darauf zuzugreifen. Der Sensorbereich muss daher von der Form eines Rechtecks mit der Breite des Roboters sein. Aria bietet zu diesem Zweck für den Zugriff auf Sensorreadings die Klasse *ArRangeBuffer* mit der Methode *getClosestBox*. Sie erhält als Parameter ein Objekt vom Typ *ArPose* das als Koordinatensystem für die Beschreibung des rechteckigen Sensorbereichs dient, also seine Ausrichtung bezüglich des Roboters beschreibt.

Die Idee ist es nun diesen Sensorbereich um den Roboter zu „drehen“ bis er kein Reading mehr enthält. Dadurch bewegt sich der Roboter immer parallel zur Wand. Ist er zu nah, entfernt er sich, ist er zu weit entfernt, nähert er sich an. Die wirkliche Ästhetik des Verfahrens liegt aber darin, dass es theoretisch mit nahezu jedem Wandverlauf klar kommt und beliebige Durchgänge und Sackgassen erkennt - durch die einfache Suche nach einem Sonarbereich **ohne** Reading.

Stärken

- einfach, effizient und allgemein (s. 3.2). Der Sonarbereich wird gedreht, enthält er kein Reading, so überträgt sich seine Ausrichtung auf den Roboter.
- automatische Anpassung an den Wandverlauf, ohne dass dieser bekannt sein muss. Es wird im Grunde genommen immer nach dem ersten möglichen Durchgang gesucht.
- Durchgänge und Sackgassen werden automatisch erkannt, Kollisionen vermieden (s. 3.3.2).

Schwächen

- Implementierung. Das Verfahren ist an Aria und dessen wenig umfangreicher Dokumentation gescheitert. Die Methode *getClosestBox* zeigt unerwünschte nicht dokumentierte Seiteneffekte. Der Sensorbereich wird zwar zunächst entsprechend dem übergebenen Koordinatensystem ausgerichtet, aber lässt sich aber nicht (durch Ändern des Bezugssystems) zuverlässig drehen. Die Stabilität des Verhaltens (s. 3.2.5) ist daher nur schwer zu beurteilen.

3.4.3 Lösungsansatz: Hough Transformation

Die Grundidee besteht darin die Wand durch eine lineare Funktion zu approximieren indem jedes Reading als Funktionswert im lokalen Koordinatensystem des Roboters interpretiert wird. Jede Gerade in der Ebene kann durch die Hesse'sche Normalform dargestellt werden. Dabei handelt es sich um ein normiertes Skalarprodukt. Für jeden Punkt (x, y) der Geraden g gilt, dass die Projektion seines Ortsvektors $\vec{x} = (x, y)^T$ auf den Normalenvektor $\vec{n}$ der Geraden konstant gleich ρ ist und damit den Abstand der Geraden vom Ursprung beschreibt.
Über den Kosinussatz lässt sich die Beziehung ableiten $\vec{x} \cdot \vec{n} = |\vec{x}| \cdot |\vec{n}| \cdot cos(\alpha)$, mit α dem Zwischenwinkel. Dabei beschreibt $\rho = |\vec{x}| \cdot cos(\alpha)$ die Projektion von $\vec{x}$ auf den Normalenvektor $\vec{n}$. Es gilt also $\rho = \vec{x} \cdot \vec{n}/|\vec{n}|$. Definiert man $\vec{n} = (cos(\theta), sin(\theta))^T$, mit θ dem Winkel des Normalenvektors, so folgt aus der trigonometrischen Beziehung $sin(\theta)^2 + cos(\theta)^2 = 1$, dass $|\vec{n}| = 1$ und damit $\rho = x \cdot cos(\theta) + y \cdot sin(\theta)$. Diese Gleichung beschreibt nun eindeutig eine Gerade in der Ebene.
Die Hough Transformation bedient sich nun dieser Gleichung. Dahinter steckt die Idee dass sich jede Gerade die durch einen Punkt (x, y) im karthesischen Koordinatensystem verläuft mittels dieser Gleichung ebenfalls als ein Punkt (θ, ρ) im sog. Hough-Raum darstellen lässt. Nun lassen sich beliebig viele Geraden durch einen einzelnen Punkt (x, y) legen. Diese Geradenschar stellt sich im Hough-Raum als sinusförmige Kurve dar und beschreibt die Menge aller möglichen Geraden durch den Punkt (x, y). Liegen nun mehrere Punkte auf einer Geraden, so schneiden sich ihre Kurven in einem Punkt der die gemeinsam Gerade durch beide Punkte beschreibt. Der Praktische Lösungsansatz besteht nun darin den Hough Raum zu diskretisieren, d.h. man betrachtet nur eine endl. Menge von Winkeln und Abständen und erhält dadurch eine Hough-Matrix (Akkumulatorfeld). Man iteriert nun also für jedes Reading über alle diskreten Werte θ, mit $-\frac{\pi}{2} < \theta \leq \frac{\pi}{2}$ und berechnet mittels obiger Formel den zugehörigen Abstandswert ρ. Anschliessend wird der Wert (θ, ρ) in der Hough Matrix um 1 erhöht. Liegen nun k versch. Punkte auf einer gemeinsamen Geraden, so akkumuliert sich an der entsprechenden Stelle (θ, ρ) in der Hough Matrix der Wert k. Der global grösste Eintrag in der Hough Matrix entspricht dabei derjenigen Geraden die durch die meisten Punkte verläuft und ist daher die beste lineare Näherung der Wand. Die Problematik des Verfahrens liegt in seinem Umgang mit Richtungsänderungen. Es liegt in der Natur des Verfahrens, dass prinzipiell alle möglichen Geraden zur Approximation der Wand berechnet werden. Tritt also eine Richtungsänderung auf, wie in Beispiel 3.5, so zeigt sich dies anhand mindestens zweier sog. Cluster in der Hough Matrix. Jedes von ihnen besitzt (im Idealfall ein eindeutiges) lokales Maximum das die entsprechende Kante approximiert. Fehlreadings und

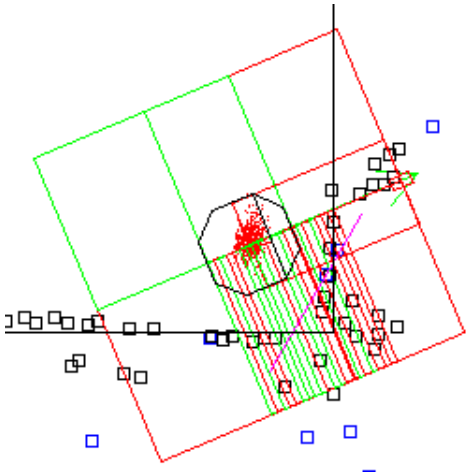


Abbildung 3.5: Konkaver Wandverlauf

Ausreisser führen jedoch dazu, dass sich meist deutlich mehr als nur zwei Cluster ausprägen. Oft ist es sogar so, dass überhaupt kein eindeutiges, globales Maximum mehr ausgemacht werden kann. Matlab beipielsweise berechnet fünf gleichwertige globale Maxima für den Wandverlauf 3.5. Um damit umgehen zu

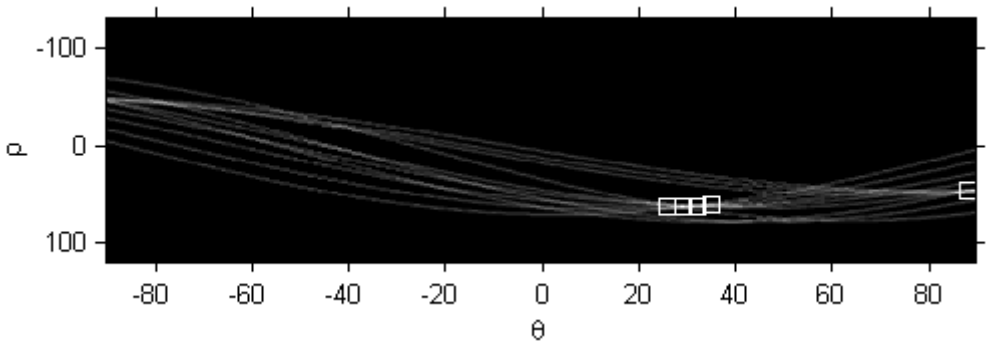


Abbildung 3.6: Durch (ρ, θ) aufgespannter Hough Raum

können muss das Verfahren also eine recht komplexe Clusteranalyse durchführen. Die Schwierigkeit dabei ist es, ohne Kenntnis der realen Situation (sie soll ja berechnet werden) eine Entscheidung über den Wandverlauf zu treffen. Unter Umständen müssen dazu auch rein lokale Maxima miteinbezogen werden. Dabei stellt sich zwangsläufig die Frage nach welchen Kriterien man Geraden auswählt, wie man sie miteinander in Beziehung setzt und gewichtet um eine möglichst exakte Vorstellung von der realen Situation zu bekommen. Das alles sind keine

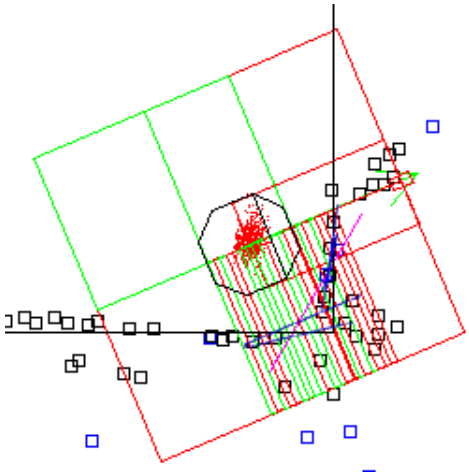


Abbildung 3.7: Approximation der Wand durch Hough Transformation

trivialen Probleme.

Stärken

- Die Approximation mittels Hough Transformation verhält sich vergleichsweise stabil gegenüber Ausreissern. Es liegt in ihrer Natur, dass sie nicht mit anderen Readings (entlang einer „Hauptrichtung“) auf einer Geraden liegen sondern davon abweichen. Dadurch werden sie in aller Regel ignoriert.

Schwächen

- Aufwand. Er steht in direktem Bezug zur Dimension der Hough Matrix. Eine Winkel- und Abstandsauflösung von 1° bzw. $1cm$ bei einer Sensorreichweite von $1m$ führt bereits zu einer 180x100 Matrix mit entsprechenden 18000 möglichen Einträgen. Dazu müssen für jedes Reading 180 Geraden berechnet werdnen.

- Clusteranalyse. Insbesondere durch Richtungsänderungen der Wand (aber auch durch Fehlreadings und Ausreisser) bilden sich in zunehmendem Masse Cluster in der Hough Matrix. Sie alle müssen auf ihre Streuung/Konzentration um ihr Zentrum, auf den Wert ihres lokalen Maximums, auf ihre Position gegenüber anderen Clustern und evtl. auf weitere Merkmale hin untersucht werden.

- Entscheidungsfunktion. Kennt man die Cluster und ihre Merkmale, so müssen sie beurteilt werden. Es ist nötig zu beurteilen in welcher Beziehung sie

zueinander stehen und wie sie anhand ihrer Merkmale gewichtet werden sollen. Ähnlich der Bildverarbeitung geht es darum ein bestimmtes Muster (Verteilung der Readings) anhand seiner Merkmale einem bestimmten Wandverlauf zuzuordnen bzw. zu klassifizieren.

- Extrapolation. Wie auch die in 4.1 gewählte Methode der kleinesten Quadrate, hat die Hough Transformation die unangenehme Eigenschaft, dass sie Geraden berechnet (von unendlicher Länge). Dadurch wird der Verlauf der Wand implizit auch extrapoliert wodurch es wiederum erschwert wird, Durchgänge oder Löcher in der Wand zu erkennen die eine Durchfahrt erlauben würden.

4 Lösung

Die implementierte Lösung baut im Wesentlichen auf zwei Algorithmen zur Approximation auf, die im Folgenden genauer beschrieben werden. Doch zunächst zum grundlegenden Aufbau. Um seine Aufgabe zu erfüllen muss das Verhalten zunächst die Wand erkennen, sie anschliessend approximieren um dann eine Entscheidung darüber zu treffen ob und in wie fern Geschwindigkeit und Ausrichtung angepasst werden sollen.
Grundsätzlich ist das Verhalten zunächst inaktiv und beobachtend. Wird eine Wand detektiert bzw. der Kontakt zu einer bereits bekannten Wand bestätigt, so werden durch eine Vielzahl feingliedriger Sensorbereiche zur entsprechenden Seite des Roboters hin zusätzliche Readings erfasst. Auf ihnen baut nun die Approximation der Wand auf.
Der Algorithmus zur tangentialen Approximation betrachtet jedes einzelne Reading $p = (x_p, y_p)$ für sich und berechnet daraus die Ausrichtung und den Abstand einer Geraden senkrecht zum Ortsvektor $\vec{p} = (x_p, y_p)^T$. Durch arithmetische Mittelung dieser Werte erhält man eine mittlere Steigung und einen mittleren Abstand.
Der Algorithmus zur Approximation mittels kleinster Quadrate Methode betrachtet anschliessend die Readings in ihrer Gesamtheit. Dabei wird versucht eine Gerade so durch die Readings zu legen, dass das Quadrat der Abweichung des approximierten Punktes vom entsprechenden Reading über alle Readings minimiert wird.
Aber wozu zwei Approximationen? Nun im Idealfall - bei einem geraden Wandverlauf - erhält der Roboter die gleiche Zahl an Readings $p_i = (x_i, y_i)$ mit $x_i < 0$ und $x_i > 0$. D.h. der Erwartungswert $E[x_i]$ liegt bei $x_i = 0$. Dann berechnen beide Verfahren einen nahezu gleichen Winkel für die Approximation der Wand und das Verhalten übernimmt die Werte der kleinsten Quadrate Methode. Nähert sich der Roboter allerdings im weitesten Sinne dem Ende einer Wand, so verlagert sich dieser Schwerpunkt zunehmend, so dass $E[x_i] < 0$ und der berechnete Winkel der tangentialen Approximation zu- bzw. abnimmt. In gleichem Masse nimmt aber auch die Diskrepanz zwischen den mit beiden Methoden berechneten Winkeln zu. Übersteigt dieser Unterschied schliesslich einen bestimmten Grenzwert, so übernimmt das Verhalten statt dessen die Werte der gemittelten tangentialen Approximation. Dadurch gelingt es dem Verhalten seitliche Durchgänge zu erkennen. Sind nun Winkel und Abstand der Wand berechnet, so wird abhängig von letzterem noch ein bestimmter Korrekturwinkel hinzu addiert um es dem Roboter zu ermöglichen sich der Wand etwas anzunähern bzw. sich von ihr zu entfernen. Im Hinblick auf die Forderung 3.2.5 nach einem stabilen Fahrverhalten wird schliesslich noch geprüft ob die Ausrichtung überhaupt korrigiert

werden soll. Unterschreitet der berechnete Winkel einen bestimmten Wert, so ändert das Verhalten die Ausrichtung nicht und der Roboter wird nicht ständig zu kleinen Korrekturen gezwungen.

4.1 Kleinste Quadrate Approximation

Die Methode der kleinsten Quadrate [Wiki05] ist das mathematische Standardverfahren zur Ausgleichsrechnung: Es ist eine Menge aus Datenpunkten gegeben, die Sonarreadings repräsentieren. Durch diese Punkte soll eine möglichst genau passende parameterabhängige Kurve gelegt werden. Dazu bestimmt man ihre Parameter numerisch, indem die Summe der quadratischen Abweichungen der Kurve von den beobachteten Punkten minimiert wird. Aus praktischen Gründen verwenden wir eine lineare Approximation mit der Geradengleichung $y = m \cdot x + b$. Wir erhalten für n verschiedene Messwerte ein Gleichungssystem, das sich wie folgt in Matrixschreibweise darstellen läßt:

$$min_{b,m} = || \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} b \\ m \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} ||_2 = min_x ||A_x - b||_2 \tag{4.1}$$

Für die resultierende Ausgleichsgerade, unsere Approximation, lassen sich die Lösungen für die Parameter direkt angeben als

$$m = \frac{\sum_{i=1}^{n} x_i y_i - n \cdot \overline{xy}}{\sum_{i=1}^{n} x_i^2 - n \cdot (\overline{x})^2} \tag{4.2}$$

$$b = \overline{y} - m\overline{x} \tag{4.3}$$

mit

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \text{ und } \overline{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{4.4}$$

als arithmetischem Mittel der x-Werte und y-Werte. Die Lösung für m kann mit Hilfe des Verschiebungssatzes auch als

$$m = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} (x_i - \overline{x})^2} \tag{4.5}$$

angegeben werden, was als Grundlage für unsere Berechnung dient. *Als Voraussetzung für dieses Verfahren gilt daher, dass mindestens zwei Readings mit*

unterschiedlichen x-Werten existieren! Die Gerade im lokalen Koordinatensystem des Roboters ist nun vollständig bestimmt wenn sowohl ihr Winkel α als auch der Punkt $p_{min} = (x_{min}, y_{min})$ der Geraden mit dem geringsten Abstand r_{min} vom Ursprung bekannt sind:

$$\alpha = atan(m) \tag{4.6}$$

$$r_{min} = cos(\alpha) \cdot b \tag{4.7}$$

$$\vec{p}_{min} = \begin{pmatrix} x_{min} \\ y_{min} \end{pmatrix} = \begin{pmatrix} sin(\alpha) \cdot r_{min} \\ cos(\alpha) \cdot r_{min} \end{pmatrix} = \begin{pmatrix} sin(\alpha) \cdot cos(\alpha) \cdot b \\ cos^2(\alpha) \cdot b \end{pmatrix} \tag{4.8}$$

Ein problematisches Verhalten zeigt sich allerdings bei konvexem Wandverlauf mit Winkeln die betragsmässig grösser sind als $\frac{\pi}{2}$. Hier wird das Ende der Wand

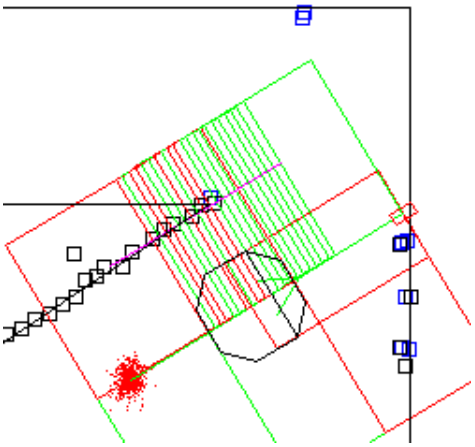


Abbildung 4.1: Kleinste Quadrate Methode bei konvexem Wandverlauf mit Winkel $|\alpha| > \frac{\pi}{2}$

bzw. ihre plötzliche Richtungsänderung nicht erkannt. Das liegt daran, dass der Algorithmus nicht die Verteilung der Readings über die Sensorbereichte betrachtet. Daher wird nicht bemerkt, dass sich die Wand nach hinten hin aus den Sensorbereichen hinaus bewegt.

4.2 tangentiale Approximation

Die Approximation mittels Tangenten soll dazu dienen die Schwächen der Methode der kleinsten Quadrate zu beheben. Dahinter steht die Überlegung, dass sich bei konvexem Wandverlauf mit betragsmässig grösseren Winkeln als $\frac{\pi}{2}$ die Zahl der Readings zunehmend verringert bis schliesslich nur noch ein einziges Reading $p = (x_p, y_p)$ zur Verfügung steht um eine Wand zu approximieren. Gehen wir wiederum von einer linearen Approximation aus, so muss es sich bei p folglich um den nächsten Punkt der Geraden handeln. Demzufolge ist der entsprechende Ortsvektor $\vec{p}$ gleichzeitig Normalenvektor der Geraden und die Gerade wiederum Tangente an einen Kreis mit Radius $r = \sqrt{x_p^2 + y_p^2}$. Um nun also den Verlauf der Geraden zu bestimmen muss nur noch ihre Steigung m bzw. ihr Winkel α berechnet werden:

$$m = -\frac{x_p}{y_p}$$

$$\alpha = \begin{cases} atan(m) & \text{falls } y =!0 \\ 90 & \text{sonst} \end{cases}$$

Wie bereits in 4.1 angesprochen liegt die Schwäche der Methode der kleinsten Quadrate darin, dass sie nur schlecht mit spitzen Winkeln bei konvexen Wandverläufen umzugehen vermag da der Algorithmus nicht bemerkt, dass die Readings langsam nach hinten aus den Sensorbereichen hinauswandern (s.4.1). Dabei ändert sich aber gleichzeitig die Verteilung der Readings. D.h. ihr „Gleichgewicht“ verlagert sich bezüglich des Ursprungs. Im Grunde genommen wandert mit den Readings selbst auch der Erwartungswert ihrer x-Koordinate von 0 zu einem Wert < 0. Die Überlegung geht nun dahin, sich dieses Ungleichgewicht nutzbar zu machen indem die oben beschriebene tangentiale Approximation über alle Readings n, (x_i, y_i) gemittelt wird.

$$\overline{\alpha} = \frac{1}{n} \cdot \sum_{i=1}^{n} atan(-\frac{x_i}{y_i}) \tag{4.9}$$

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{4.10}$$

$$\overline{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{4.11}$$

Damit gelingt es dem Verhalten seitliche Durchgänge in weitesten Sinne wahrzunehmen.

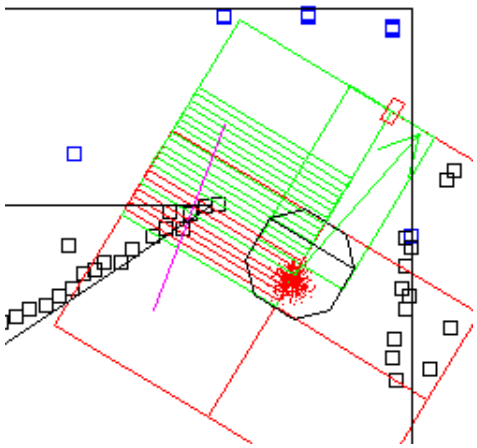


Abbildung 4.2: gemittelte tangentiale Approximation bei konvexem Wandverlauf mit Winkel $|\alpha| > \frac{\pi}{2}$

5 Implementierung

Aus praktischen Überlegungen gliedert sich die Implementierung der Wandverfolgung in zwei Klassen: die Hilfsklasse *SfSensorArea* die zur Verwaltung eines rechteckigen Sensorbereichs dient und das eigentliche Behaviour *SfWallFollow* mit der entsprechenden *fire()* Methode.
Der wichtigste Baustein unseres Programm ist die Funktion *approximate()* der Klasse *SfWallFollow*, mit der versucht wird den Wandverlauf anhand von Readings zu approximieren. Sie erhält einerseits einen Menge von Readings als Array von *ArPose* Objekten und andererseits einen Wert der ihre Anzahl beschreibt. Der Rumpf der Methode gliedert sich in zwei Teile auf, die zum Einen die tangentiale Approximation und zum Anderen die Approximation mittels kleinster Quadrate Methode implementiert.

Listing 1: Die Approximation

```
1  ArPose* SfWallFollow::approximate(ArPose** s, int n)
2  {
3     /* nur ein reading, tangentiale Approximation */
4     if(n == 1)
5     {
6        double x = s[0]->getX();
7        double y = s[0]->getY();
8        double a = 0.0;
9        ((y > eps) || (y < -eps)) ? a = 180/M_PI*atan(-x/y) : a = wall*90;
10       return new ArPose(x, y, a);
11    }
12    /* mehrere readings, kleinste Quatrate Methode */
13    else if(n >= 2)
14    {
15       /* berechnung des durchnittes der x and y werte */
16       double avg_x = 0.0;
17       double avg_y = 0.0;
18       for(int i=0; i<n; i++)
19       {
20          avg_x += s[i]->getX();
21          avg_y += s[i]->getY();
22       }
23       avg_x = avg_x /n;
24       avg_y = avg_y /n;
25
26       /* berechnung der tangetialen approximation mit y = m*x+b:
27       m = sum[(x(i) - avg_x)*(y(i) - avg_y)]/sum[(x(j) - avg_x)^2]
28       and b = avg_y - m*avg_x */
29       double m = 0.0;
30       double b = 0.0;
31       double a = 0.0;
32       double t = 0.0;
33       for(int j=0; j<n; j++)
34       {
35          m += (s[j]->getX() -avg_x)*(s[j]->getY() -avg_y);
36          t += (s[j]->getX() -avg_x)*(s[j]->getX() -avg_x);
37       }
38       (t > eps || t < -eps) ? m = m/t : m = 0.0;
39       (m > eps || m < -eps) ? a = atan(m) : a = 0.0;
```

```
40          b = avg_y - m*avg_x;
41          return new ArPose(-sin(a)*cos(a)*b, cos(a)*cos(a)*b, 180/M_PI*a);
42      }
43
44      /* keine readings */
45      return NULL;
46  }
```

Welche Art der Approximation gewählt wird, entscheidet sich anhand des 2. Parameters des Aufrufes. Dieser Parameter enthält die Zahl der Readings.

Listing 2: Aufruf der tangentiale Approximation

```
1   (...)
2   /* fuer jedes Reading wird einzeln die tangentiale Approximation
3   der Wand berechnet und die Werte werden aufsummiert. */
4   for(int i=0; i<sensor_sub; i++)
5   {
6       sensork[i]->setArea(s, 0, s+w, wall*sonar_range );
7       sensork[i]->update();
8       atm_approx = approximate( &(sensork[i]), 1);
9       if(atm_approx != NULL)
10      {
11          sum_a += atm_approx->getTh();
12          sum_x += atm_approx->getX();
13          sum_y += atm_approx->getY();
14          c++;
15      }
16      delete atm_approx;
17      s += w;
18      }
19  }
20  /* Mittelung von Winkel und Koordinaten */
21  atm_approx = new ArPose(sum_x/c, sum_y/c, sum_a/c);
22  (...)
```

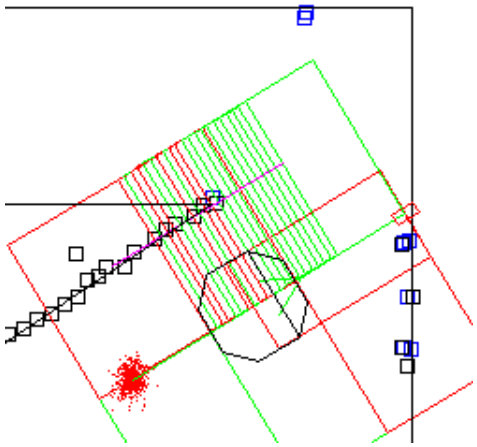
Listing 3: Aufruf der Least Square Approximation

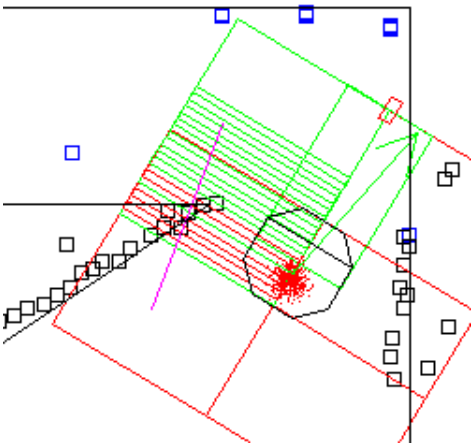
```
1   (...)
2   /* Least Square Methode: approximate wird mit den Readings
3   aller Sensorbereiche aufgerufen*/
4   lsm_approx = approximate(sensork, sensor_sub);
5   (...)
```

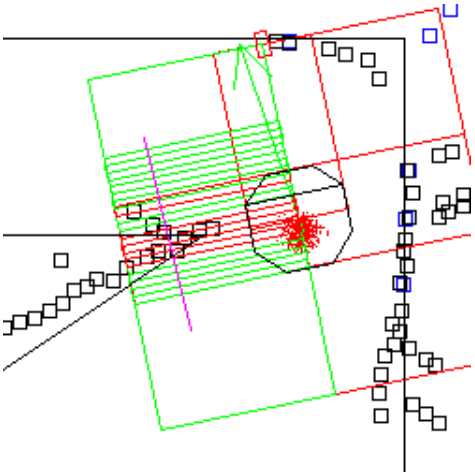
6 Ergebnis

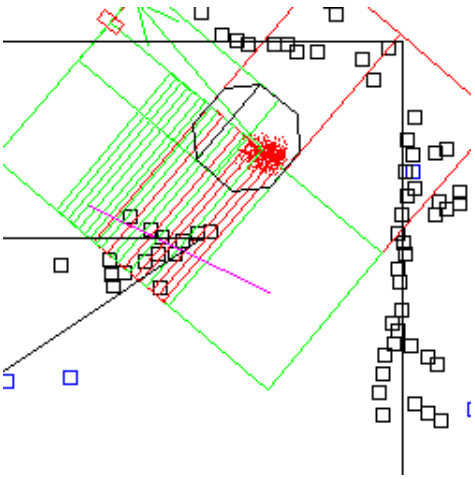
Und funktioniert das ganze? Eine berechtigte Frage. Wir wollen nun also untersuchen in wie fern die in 3.2 formulierten Forderungen nach einer stabilen (3.2.5) und algemeinen (3.2.4) Lösung erfüllt werden. Dazu veranschaulichen wir uns das Verhalten anhand verschiedener Wandverläufe.

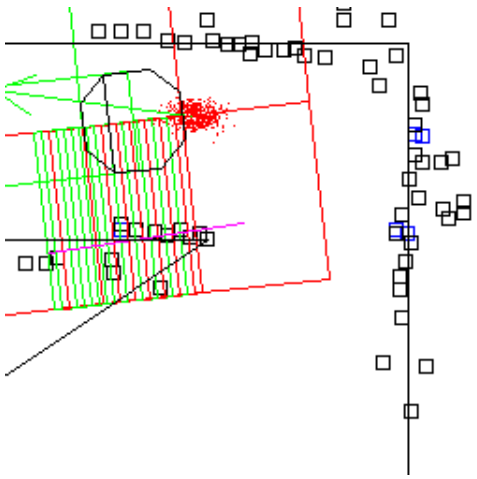
6.1 allgemeiner konvexer Wandverlauf (135°)



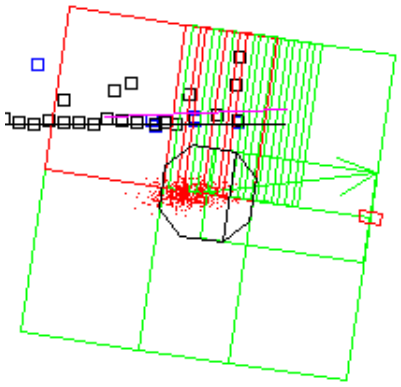


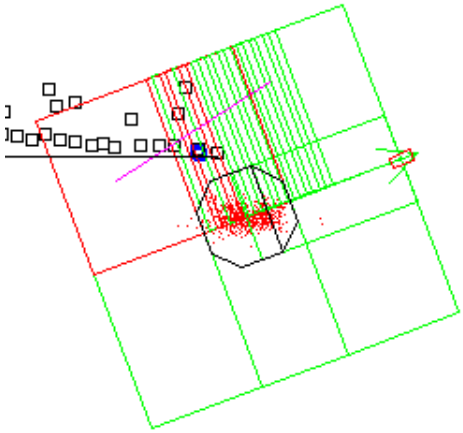


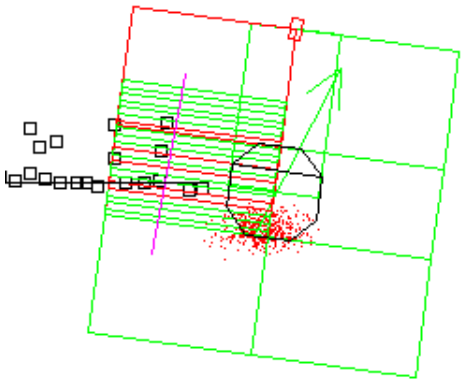


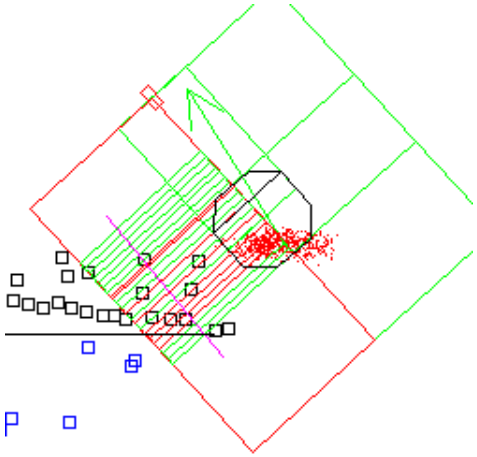


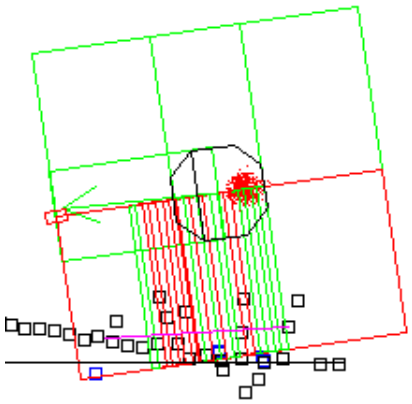
6.2 spezieller konvexer Wandverlauf (180°)



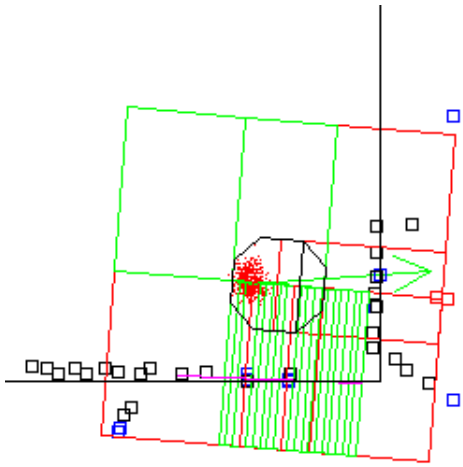


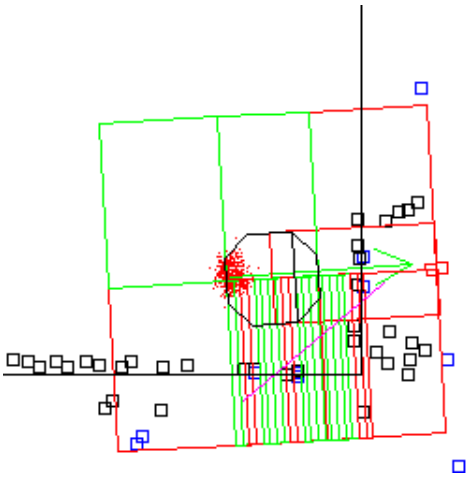


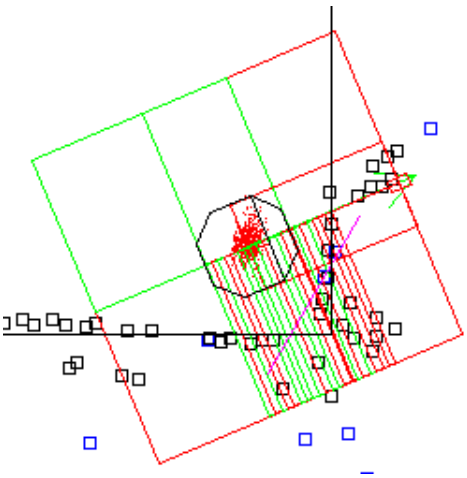


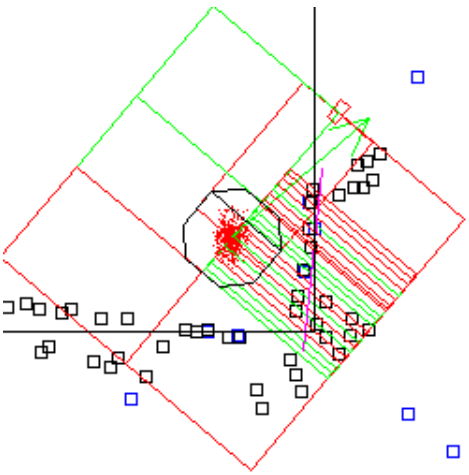


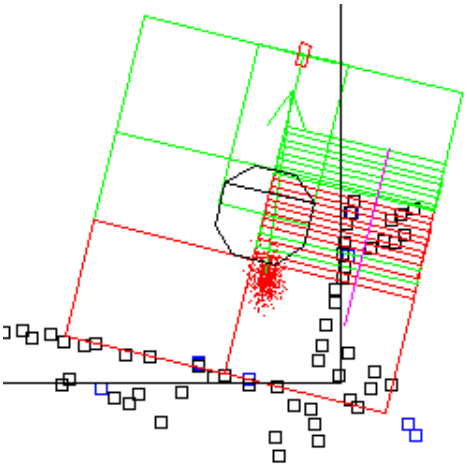
6.3 allgemeiner konkaver Wandverlauf (90°)



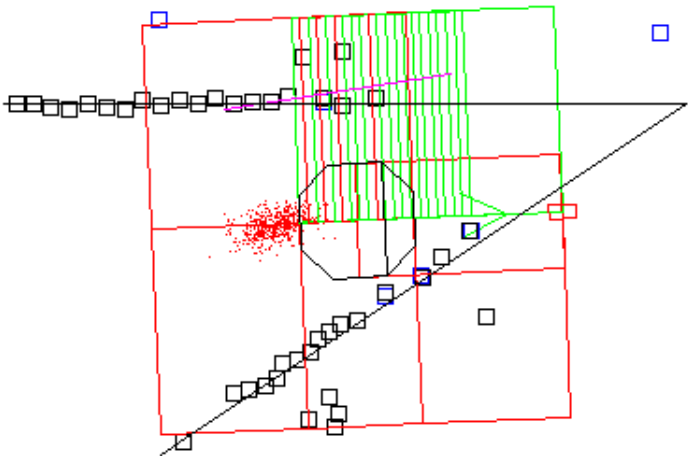


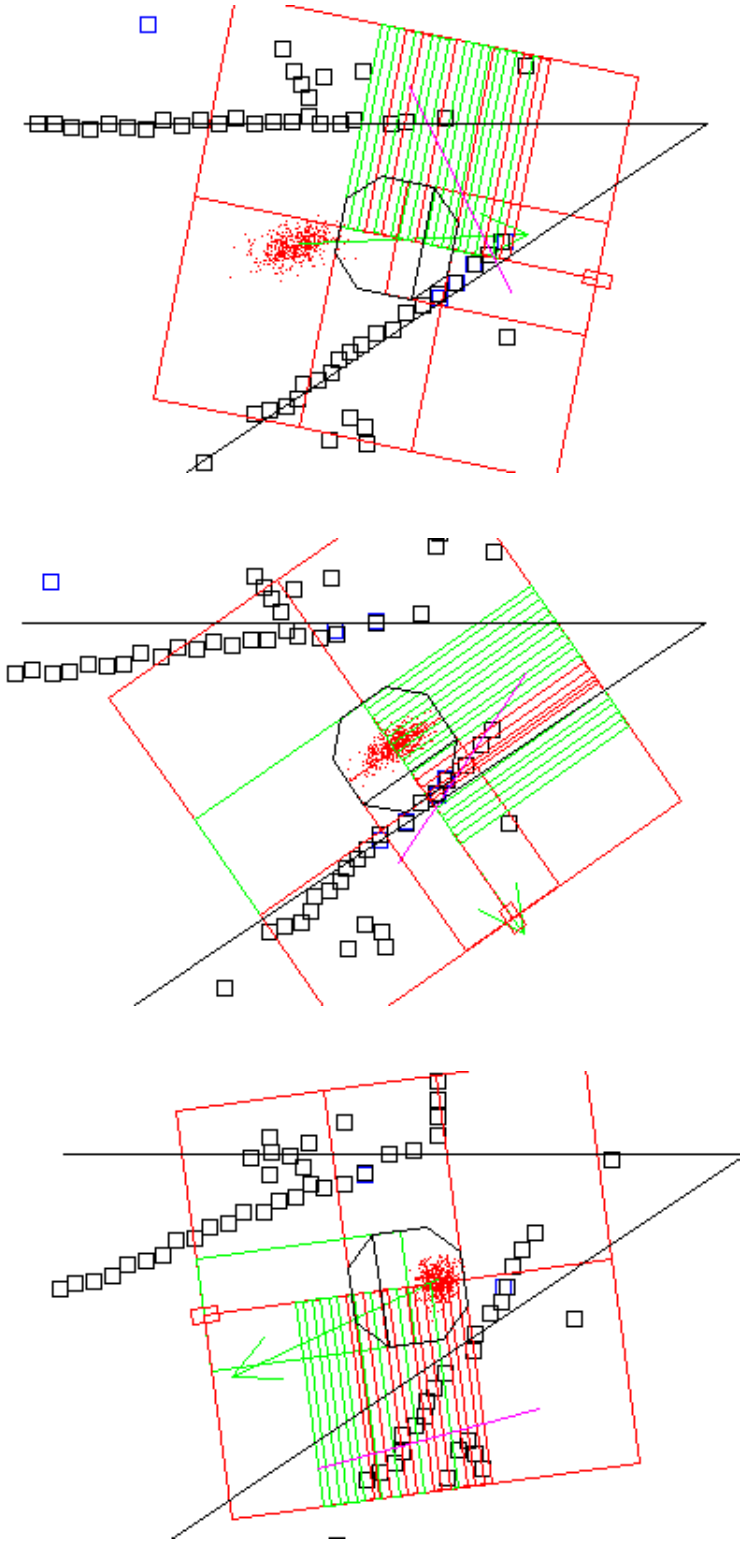


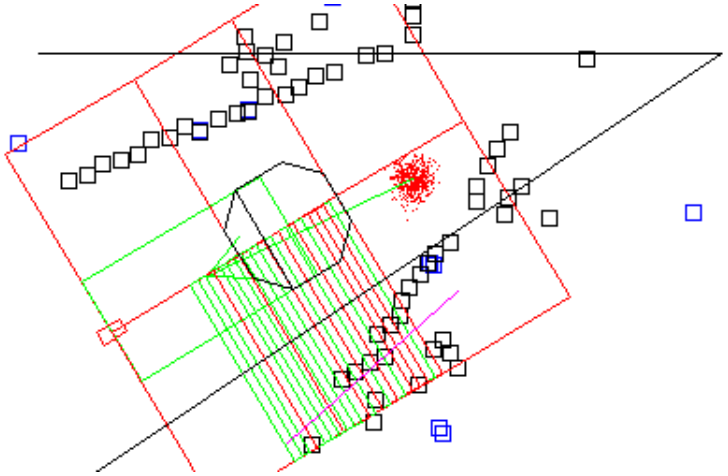




6.4 allgemeiner konkaver Wandverlauf (-135°)







6.5 Der ultimative Elchtest

Der „Elchtest“ beschreibt ein Weltmodell das die Lösung mit einer Reihe von Problemfällen konfrontiert. Allerdings werden die verschiedenen Situationen nicht isoliert betrachtet, sondern gehen fliessend ineinander über. Insbesondere geht es darum die Robustheit bezüglich verschiedener Wandverläufe und ihrer Übergänge zu testen. Es wird eine Raumumgebung mit Durchgängen und Wänden simuliert, eine „Ruckelpiste“ soll prüfen ob und in wie fern sich simulierte Ausreisser auswirken und ob das Verhalten tatsächlich schnell und sicher auf Änderungen reagiert. Umfangreiche Testläufe haben gezeigt, dass das Verhalten in der Tat die in 3.2 gewünschten Anforderungen erfüllt.

6.6 Schwächen

- Partielle Blindheit. Der Roboter beobachtet nahezu ausschliesslich die Wand und die gegenüberliegende Seite wird völlig ignoriert. Nähert sich also von dort ein Hindernis, etwa bei der Passage einer engen Durchfahrt oder bei der Verengung eines Flurs, so bremst er lediglich ab. Dadurch kann er sich gewissermassen festfahren. Allerdings ist es auch eine Ermessensfrage in wie fern eine Hinderniserkennung von der Wandverfolgung implementiert werden soll. Trotz unserer bewussten Entscheidung gegen die Implementation entsprechender Verhaltensweisen soll hier nochmals ausdrücklich auf diese Schwäche hingewiesen werden.

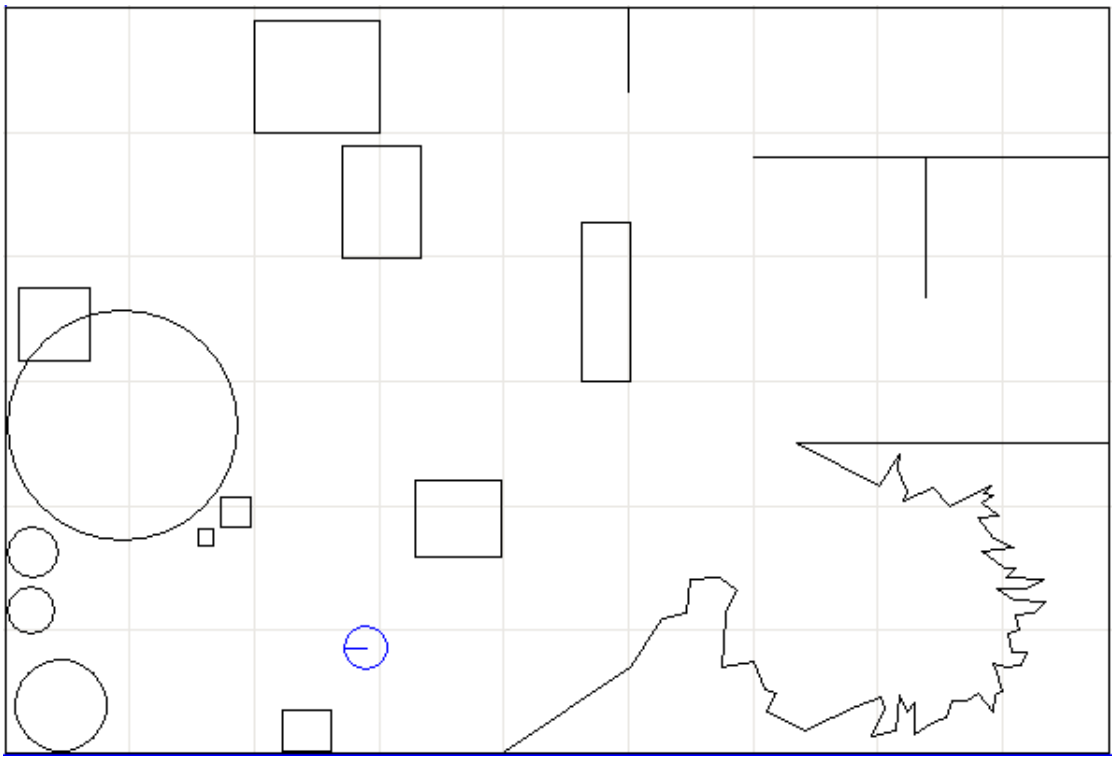


Abbildung 6.1: Elchtest zur Wandverfolgung

- Im Falle eines konvexen Wandverlaufs mit einer Richtungsänderung von 180° (s.6.2) tendiert das Verhalten dazu sich bis auf die maximale Sonarreichweite von der Wand zu entfernen.
- Im Falle eines konkaven Wandverlaufs mit Richtungsänderungen $> 135°$ bzw $< -135°$, also bei Sackgassen, kann es vorkommen, dass der Roboter wie bereits angesprochen die gegenüberliegende Wand nicht wahrnimmt und entsprechend hängen bleibt. Eine mögliche Lösung des Problems wäre es den Abstand zum frontalen Hindernis abzufragen und wenn nötig die Ausrichtung zusätzlich zum berechneten Wert zu ändern.

Literaturverzeichnis

[Gem05] Gemmar, Peter (2004): *Vorlesung Robotik WS 04/05*, Vorlesungsskript, FH-Trier, Seite 22-27

[SiKl03] Simon, Dirk / Kloss, Bernhard (2003): *Diplomarbeit*, Bildgestützte Navigation und Handhabung von Gegenständen mit einem autonomen Roboter, FH-Trier, Seite 14-23

[Wiki05] Wikipedia.org (Lezte Änderung: 08. Jan. 2005) Methode der kleinsten Quatrate