



# Robotik Zusammenfassung

Prof. Schiedermeier

SS 2019

Robin Atherton

# Contents

<b>1</b>	<b>Geschichte</b>	<b>1</b>
1.1	Industrieroboter . . . . .	1
1.2	Serviceroboter . . . . .	1
1.2.1	Definition . . . . .	1
1.2.2	Klassen . . . . .	2
<b>2</b>	<b>Software-Architekturen für mobile Robotersysteme</b>	<b>3</b>
2.1	Probleme und Anforderungen . . . . .	3
2.1.1	Definition mobile Roboter . . . . .	3
2.1.2	Umgebung mobiler Roboter . . . . .	3
2.1.3	Roboterkontroll-Architekturen . . . . .	4
2.1.4	Anforderungen an das Kontrollsystem eines autonomen Roboters . . . .	4
2.2	Mögliche Modelle . . . . .	5
2.2.1	Klassisches Modell - der funktionale Ansatz . . . . .	5
2.2.2	Verhaltensbasiertes Modell . . . . .	6
2.2.3	Hybrider Ansatz . . . . .	7
2.2.4	Probabilistische Robotik . . . . .	8
2.2.5	Subsumption-Architektur in Bezug auf die Anforderungen des Robot- ersteuerungssystems . . . . .	8
2.3	ROS - Robot Operating System . . . . .	9
2.3.1	Entwicklung . . . . .	9
2.3.2	Design Prinzipien . . . . .	10
2.3.3	Publish - Subscribe . . . . .	11
2.3.4	Parameter Server und Konfigurationsdateien . . . . .	12
<b>3</b>	<b>Lokalisation autonomer mobiler Robotersysteme</b>	<b>13</b>
3.1	Abgrenzung: Lokalisation - Mapping - SLAM - Navigation . . . . .	13
3.2	Varianten der Selbstlokalisierung . . . . .	13

## Contents

3.3	Relative Lokalisierung versus Absolute Lokalisierung . . . . .	14
3.4	Transformation von Koordinatensystemen lokale $\leftrightarrow$ globale . . . . .	15
3.5	Karten für statistische und dynamische Umgebungen . . . . .	16
3.5.1	Mapping Methoden . . . . .	17
3.5.2	Arten von Modellen . . . . .	17
3.5.3	Kontinuierliche Metrische Karten . . . . .	18
3.5.4	Grid Maps - Rasterkarten . . . . .	18
3.5.5	Adaptive Unterteilung . . . . .	20
3.5.6	Weitere Beispiele für Umgebungskarten . . . . .	21
3.5.7	Topologische Karten . . . . .	21
3.5.8	Hybrid Maps . . . . .	22
3.6	Passive und Aktive Selbstlokalisierung . . . . .	22
3.6.1	Passive Verfahren . . . . .	22
3.6.2	Aktive Verfahren . . . . .	22
3.7	Landmarken . . . . .	23
3.7.1	Definition . . . . .	23
3.7.2	Natürliche Landmarken . . . . .	23
3.7.3	Künstliche Landmarken . . . . .	23
<b>4</b>	<b>Fortbewegung, Lokalisierungsalgorithmen</b>	<b>24</b>
4.1	Relative Lokalisierung . . . . .	24
4.1.1	Dead Reckoning . . . . .	24
4.1.2	Odometrie . . . . .	25
4.1.3	2D-Scanmatching . . . . .	26
<b>5</b>	<b>Navigation</b>	<b>29</b>
5.1	Bekanntes vs. unbekanntes Terrain . . . . .	29
5.2	Navigation in unbekanntem Terrain . . . . .	29
5.2.1	Konturverfolgung . . . . .	29
5.2.2	Sensorbasierte Planer - Navigation mit Hinderniskontakt . . . . .	30
5.2.3	Labyrinth . . . . .	33
5.3	Pfadplanung für mobile Roboter in bekanntem Terrain . . . . .	35
5.3.1	Bewegungsplan für mobile Roboter . . . . .	35
5.3.2	Konfigurationsraum . . . . .	36
5.4	Algorithmen und Methoden . . . . .	37
5.4.1	Dijkstra . . . . .	38

## *Contents*

5.4.2	A* . . . . .	38
5.4.3	Wegsuche und Umgehung bekannter Hindernisse mit dem Sichtgraph- Algorithmus . . . . .	38
5.4.4	Voronoi-Diagramme . . . . .	42
5.4.5	Navigation in einer Rasterkarte . . . . .	44
5.4.6	PotentialFeldmethode . . . . .	46
<b>List of Figures</b>		<b>48</b>
<b>Listings</b>		<b>49</b>

# 1 Geschichte

## 1.1 Industrieroboter

Nach Definition der VDI-Richtlinie 2860 sind Industrieroboter universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkel frei programmierbar und sensorgeführt sind.

- Zeichnen sich durch **Schnelligkeit**, **Genauigkeit**, **Robustheit** und eine hohe **Traglast** aus.
- Einsatzgebiete: Schweißen, Kleben, Schneide, Lackieren

Zunehmend **kollaborative** Roboter, Cobots:

- Industrieroboter, die mit Menschen gemeinsam arbeiten
- Nicht mehr durch Schutzeinrichtungen im Produktionsprozess von Menschen getrennt
- Nimmt Menschen wahr, verursacht keine Verletzungen

## 1.2 Serviceroboter

### 1.2.1 Definition

- Ein **Serviceroboter** ist eine **frei programmierbare Bewegungseinrichtung**, die **teil- oder vollautomatisch** Dienstleistungen verrichtet.
- **Dienstleistungen** sind dabei Tätigkeiten, die nicht der direkten industriellen ERzeugung von Sachgütern, sondern der Verrichtung von **Leistungen für Menschen und Einrichtungen** dienen.
- Einteilung in zwei Klassen

### **1.2.2 Klassen**

- Roboter, die für professionellen Einsatzbereich: **Rettung, Landwirtschaft, Medizin**
- Roboter für den Privaten gebrauch: **Staubsauger, Rasenmäher, Pfleger**

## 2 Software-Architekturen für mobile Robotersysteme

### 2.1 Probleme und Anforderungen

#### 2.1.1 Definition mobile Roboter

‘Unter einem Roboter verstehen wir eine frei programmierbare Maschine, die auf Basis von Umgebungssensordaten in geschlossener Regelung in Umgebungen agiert, die zur Zeit der Programmierung nicht genau bekannt und/oder dynamisch und oder nicht vollständig erfassbar sind.’ ⇒ **Joachim Herzberg, Mobile Roboter**

#### 2.1.2 Umgebung mobiler Roboter

Bei **mobilen Robotern** ist die Umgebung im Detail **nicht bekannt und generell nicht kontrollierbar**

- Alle Aktionen sind von der aktuellen Umgebung abhängig
- Details sind erst zum Zeitpunkt der Ausführung der Aktionen bekannt
- Mobile Roboter müssen in einer geschlossenen Regelung
  - die Umgebung mit Sensoren erfassen
  - die Daten auswerten
  - Aktionen daraus planen
  - Aktionen mittels Koordination der Aktuatoren umsetzen

### 2.1.3 Roboterkontroll-Architekturen

#### Herausforderungen

- Robotersystem besteht aus den Gebieten **Wahrnehmung**, **Planung** und **Handlung**
- Herausforderungen an eine Roboterkontroll-Architektur, sie muss:
  - Sensorwerte erfassen und auswerten
  - Pfade planen
  - Hindernisse vermeiden
  - Komplexe Algorithmen in langen Zeitzyklen ausführen

#### Probleme bei der Software-Erstellung zur Roboterkontrolle

- Roboter sind eingebettete Systeme, die in geschlossener Regelung laufen und die Sensorströme in **Echtzeit verarbeiten** müssen
- Unterschiedliche Aufgaben → Unterschiedliche Zeitzyklen
- Unterschiedlicher Zeitskalen → kein standardisierter Kontroll- oder Datenfluss den die Architektur abbilden könnten
- Für etliche algorithmische Teilprobleme sind **keine effizienten Verfahren** bekannt
- **Prozessorkapazität ist begrenzt**

### 2.1.4 Anforderungen an das Kontrollsystem eines autonomen Roboters

#### Robustheit

- Die Umgebung des Systems kann sich ständig ändern
- Auf eine Umgebungsänderung sollte der Roboter sinnvoll reagieren und nicht verwirrt stehen bleiben.
- Verwendete Modelle der Umgebung sind ungenau.



## 2.2 Mögliche Modelle

### Unterschiedliche Ziele

- Der Roboter verfolgt zu einem Zeitpunkt eventuell Ziele, die im Konflikt zueinander stehen.
- **Beispiel:** der Roboter soll ein bestimmtes Ziel ansteuern, dabei aber Hindernissen ausweichen.

### Sensorwerte von mehreren Sensoren

- Sensordaten können verrauscht sein
- Sensoren können fehlerhafte oder inkonsistente Messwerte liefern, weil der Sensor z.B. außerhalb seines Bereichs misst für den er zuständig ist und dies nicht überprüfen kann.

### Erweiterbarkeit

- Wenn der Roboter neue Sensoren erhält, sollte dies leicht in das Programm integriert werden können.

## 2.2 Mögliche Modelle

### 2.2.1 Klassisches Modell - der funktionale Ansatz

Das **klassische Modell** wird auch als hierarchisches Model oder funktionales Model bezeichnet. Ist ein Top-Down Ansatz, besteht aus drei Abstraktionsebenen

- Die unterste Ebene: **Pilot**
- Mittlere Eben: **Navigator**
- Oberste Ebene: **Planer**

**Sense-Think-Act-Cycle** oder **SMPA** (Sense - Model - Plan - Act).

- Sensordaten, die vom Fahrzeug geliefert werden, werden in den zwei unteren Ebenen vorverarbeitet.
- Konstruktion oder Aktualisierung eines Weltmodells
- **Planer** ist die Basis aller Entscheidungen basieren auf dem zugrundeliegenden Weltmodell

## 2.2 Mögliche Modelle

- Tatsächliche Fahrbefehle werden durch unterste Ebene ausgeführt

Zyklus wird ständig wiederholt  $\Rightarrow$  wenn alle Ebenen richtig funktionieren resultiert daraus ein intelligentes Verhalten und die Erfüllung der Aufgabe.

### Nachteile

- **Sequentieller Ansatz, lange Kontrollzykluszeit**
- Gesamtsystem anfällig,  $\Rightarrow$  fällt ein Modul scheitert das Gesamtsystem
- Die Repräsentation der Umgebung muss alle notwendigen Informationen enthalten, damit ein Plan entwickelt werden kann. Planer hat nur Zugriff auf das Weltmodell  $\Rightarrow$  während Planer aktionen ausarbeitet, könnte sich die Umwelt schon wieder geändert haben.

### 2.2.2 Verhaltensbasiertes Modell

**Grundlegender Gedanke** Intelligentes Verhalten wird nicht durch komplexe, monolithische Kontrollstrukturen erzeugt, sondern durch das Zusammenführen der richtigen einfachen Verhalten und deren Interaktion.

### Definition

- Engere Verbindung zwischen **Wahrnehmung** und **Aktion**
- Jede **Roboterfunktionalität** wird in einem **Behavior** gekapselt
- Alle **Behaviors** werden **parallel ausgeführt**
- Jedes Behavior Modul operiert unabhängig von den anderen
- Alle Behaviors können auf alle Fahrzeugsensoren zugreifen und gewissermaßen die Aktuatoren ansteuern.

### Beispiel Subsumption Architektur nach Brooks

### 2.2.3 Hybrider Ansatz

- Nutzt die Vorteile der **Subsumption Architektur** und der **SMPA-Architektur**
- Der verhaltensbasierte Anteil ist nicht geeignet, auf längere Sicht zielgerichtet Aktionen zu koordinieren  $\Rightarrow$  SMPA-Anteil

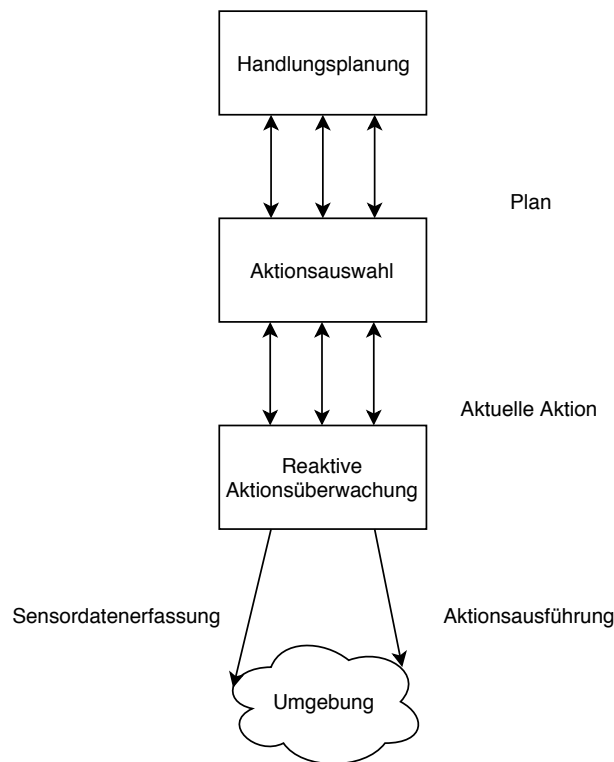


Figure 2.1: Schema der Hybridmodell Schichten

- Die **Handlungsplanung** arbeitet auf hoher, strategischer Stufe in langen Zeitzyklen
- Die **reaktive Aktionsüberwachung** enthält die Verhaltensbausteine auf operativer Ebene, die in schnellen Zeitzyklen die physische Roboteraktion anstoßen und überwachen
- die **mittlere Kontrollebene** hat die taktische Aufgabe, die jeweils **nächste Aktion aus dem Plan auszusuchen** zu instanzieren und auf die Ebene der Verhaltensbausteine zu zerlegen. Des weiteren muss die Rückmeldung von der Aktionsüberwachung interpretieren und entscheiden ob eine Aktion erfolgreich abgeschlossen ist.  $\Rightarrow$  entscheiden ob die Handlungsplanung einen anderen Plan erstellen muss

## 2.2 Mögliche Modelle

### Kritik

- Mittlere Komponente benötigt den größten konzeptuellen und programmiertechnischen Aufwand
- Das mittlere Teilproblem ist deutlich komplexer als die beiden anderen

### 2.2.4 Probabilistische Robotik

- Probabilistische Robotik berücksichtigt die **Unsicherheit der Wahrnehmung und der Aktionen**
- **Schlüsselidee** Information in Form von Wahrscheinlichkeitsdichten repräsentieren
- Eine Lokalisierung der Roboter wird unter Verwendung von Wahrscheinlichkeitstheorie oder einer Wahrscheinlichkeitsverteilung eine Aussage über die Umgebung treffen
- **Probabilistische Wahrnehmung:** wenn man Sensorwerte schätzen kann, dann kann man mit Wahrscheinlichkeitstheorie/Wahrscheinlichkeitsverteilung eine Aussage über die Umgebung treffen
- **Probabilistisches Handeln:** aufgrund der Unsicherheit über die Umgebung ist auch das Handeln mit Unsicherheit behaftet. Mit probabilistischen Ansätzen besteht die Möglichkeit Entscheidungen trotz Unsicherheit zu treffen

**Vorteil** probabilistische Verfahren können auch mit weniger präzisen Umgebungsmodellen angewandt werden.

**Nachteil** weniger effizient wegen komplexer Berechnungen, Approximation erforderlich

### 2.2.5 Subsumption-Architektur in Bezug auf die Anforderungen des Robotersteuerungssystems

#### Robustheit

- Wenn einige Steuerungsmodule ausfallen, arbeiten bei der Subsumption-Architektur in die restlichen Schichten einwandfrei ⇒ **eingeschränktes, aber sinnvolles Verhalten möglich**

### Unterschiedliche Ziele

- Mehrere Teilsituation können verschiedene Verhaltenselemente sinnvoll machen, die sich widersprechen können.
- Die Wichtigkeit einer Handlung hängt vom Kontext ab, d.h. höhere Ziele können niederere Ziele ersetzen.
- Alle zu einem Zeitpunkt möglichen Verhaltenselemente werden parallel bearbeitet.
- Das **resultierende Verhalten wird in Abhängigkeit von Umwelteinflüssen dynamisch** bestimmt
- Das Gesamtergebnis hängt nicht von einer übergeordneten Instanz ab

### Sensorwerte von mehreren Sensoren

- Der Roboter muss auch bei inkonsistenten Informationen eine Entscheidung fällen
- Die Subsumption-Architektur sieht keine zentrale Verarbeitung und Speicherung der Umweltdaten
- Jedes Modul reagiert nur auf die Daten einzelner Sensoren, es muss **kein konsistentes Abbild der Umwelt erschaffen werden**

**Erweiterbarkeit** Das bestehende Verhalten kann jederzeit durch Hinzufügen weiterer Schichten um komplexere Funktionen erweitert werden

## 2.3 ROS - Robot Operating System

### 2.3.1 Entwicklung

- **Das Architekturschema für Roboterkontrollsoftware** gibt es nicht  $\Rightarrow$  deshalb heute auch Unterstützung der Roboter-Softwareentwicklung durch Middleware wie ROS
- **Zweck:** soll die Entwicklung von Software für Roboter vereinfachen und wiederkehrende Aufgaben standardisieren
- Standard für Roboterkontrollsoftware

### 2.3.2 Design Prinzipien

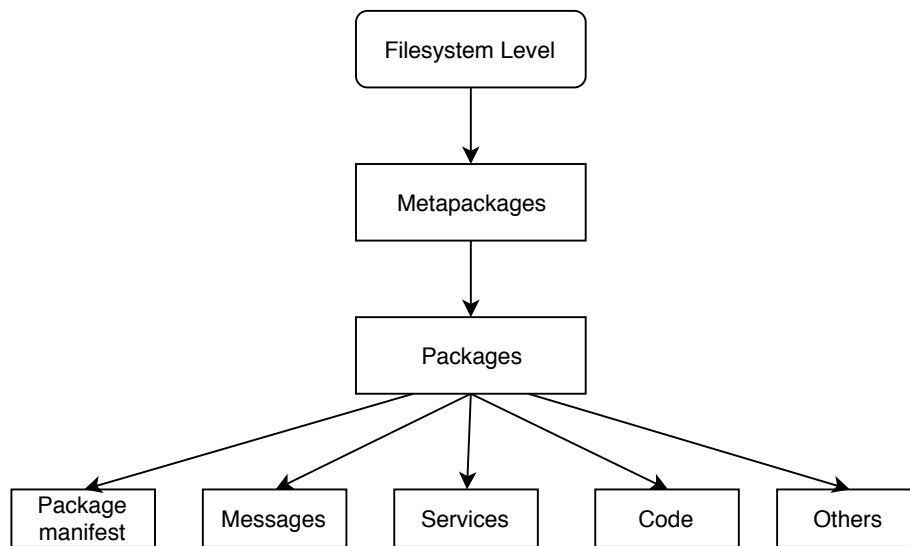


Figure 2.2: Filesystem, das ROS zugrunde liegt

- Ein Package beinhaltet die ROS Prozesse, welche auch Nodes genannt werden
- Komplexe Prozesse werden durch Netzwerke von Nodes bewerkstelligt
- Roboterkontrollprogramm besteht aus vielen Prozessen, die potentiell über mehrere Rechner verteilt sein können
- Wichtigster Knoten ist der Master  $\Rightarrow$  Abwicklung der internen Kommunikation
- Andere Knoten können nur starten, wenn ein Master existiert
- Nodes müssen sich beim Master anmelden
- Funktionalität (Kommandos, Ausführung v. Algorithmen) wird in eigenen Nodes realisiert
- Nodes sind in verschiedene Prozesse getrennt  $\Rightarrow$  fehlerhafte Knoten hat i.d. Regel wenig Auswirkungen auf die anderen
- Knoten werden über Publish-Subscribe verknüpft
- **Asynchrone Nachrichten** werden durch Topics ausgetauscht
- **Synchrone Nachrichten** werden durch Services ausgetauscht

### 2.3.3 Publish - Subscribe

**Nodes** sind Software-Module, die die Verarbeitung durchführen. Sie kommunizieren über Topics miteinander und tauschen dabei Nachrichten.

**Kommunikation** Die Kommunikation basiert auf einem **Publish Subscribe Pattern**

- Wenn Daten weitergegeben werden sollen, wird ein Publisher erzeugt
- Publisher registriert sich beim Master und gibt Topics an
- Daten können in anderen Knoten abgerufen werden – dazu wird ein (oder mehrere) Subscriber angelegt
- Subscriber fragt beim Master bezüglich gewünschten Topics an
- Daten werden über TCP/IP Sockets übertragen

#### Topics

- Themen, zu denen die Nodes Messages versenden
- Topics sind einfach Strings
- Verschiedene Nodes können zu einem bestimmten Topic Nachrichten versenden
- Ein Node kann sich prinzipiell zu mehreren Topics einschreiben und mehrere Topics publizieren

#### Services

- Nachteil von Publish Subscribe wird durch Services geschlossen
- Sind eine weitere Art, wie Nodes kommunizieren können
- Synchroner Nachrichtenaustausch mithilfe von Requests, welche von anderen Nodes mit einer Response beantwortet werden
- Ein Knoten registriert eine Aktion (Service), namentlich beim Master
- Ein Service Caller kann die Ausführung eines Services anstoßen, sobald dieser verfügbar ist
- geeignet für RMI oder einmalige Anfragen

## 2.3 ROS - Robot Operating System

### Messages

- werden von Nodes bei der Kommunikation
- Messages sind streng typisierte, möglicherweise verschachtelte Datenstrukturen, die aus den primitiven Typen int, float, bool bestehen
- Eine Message kann andere Messages oder Felder von Messages enthalten

### 2.3.4 Parameter Server und Konfigurationsdateien

- Der ParameterServer auf dem Master Knoten enthält eine Art Wörterbuch für Werte.
- Alle Ressourcen wie Knoten, Nachrichten oder Parameter existieren in einer hierarchischen Namensstruktur.
- Speichert z.B. Konfigurationsdateien



# 3 Lokalisation autonomer mobiler Robotersysteme

## 3.1 Abgrenzung: Lokalisation - Mapping - SLAM - Navigation

- **Lokalisation** Ermitteln der aktuellen Position des Roboters.
- **Kartenerstellung, Mapping oder Umgebungsmodellierung** hilft bei Entscheidung  
Kartenerstellung bedeutet die Auswertung der vom Roboter mittels Sensoren erfassten Daten der Umgebung mit dem Ziel, ein Umgebungsmodell zu erzeugen oder zu vervollständigen.
- **Großes Problem Mapping**
- **Pfadplanung oder Navigation** beantwortet die Frage **Wie gelange ich dorthin?**  
Bewegungsplanung oder Pfadplanung bedeutet die Berechnung der Fahrroute und der daraus abgeleiteten Bahn vom aktuellen Punkt zum Zielpunkt
- Man unterscheidet zwischen Navigation in unbekannter und in bekannter Umgebung.
- Selbstlokalisierung und Kartenerstellung bedingen sich gegenseitig.

## 3.2 Varianten der Selbstlokalisierung

### Lokale Selbstlokalisierung (position tracking)

- Die Startposition des Roboters ist ungefähr bekannt.
- Es handelt sich um **relative** Selbstlokalisierung

### 3.3 Relative Lokalisierung versus Absolute Lokalisierung

- Sobald sich der Roboter bewegt, muss aufgrund neuer Sensordaten die Position neu berechnet werden.
- Bezugspunkt ist der Startpunkt.
- **Methoden** Odometrie und Trägheitsnavigation

#### Globale Selbstlokalisierung

- Die Startposition ist unbekannt.
- Es handelt sich um **absolute Positionierung**
- An welcher Position der Roboter befindet, entscheidet er durch Auswerten seiner Sensordaten und durch erkennen von **signifikanten** Umgebungsmerkmalen
- **Mögliche Methode:** Triangulation

#### Kidnapped Robot Problem

- Die Position des Roboters ist anfangs bekannt
- Der Roboter wird willkürlich mit temporär deaktivierten Sensoren an eine beliebige andere Position versetzt, ohne darüber informiert zu werden.
- Auch dann muss das Verfahren robust die Position wiederfinden, zunächst muss der Roboter dies erkennen und sich dann relokalisieren
- Es muss eine erneute globale Lokalisierung durchgeführt werden

## 3.3 Relative Lokalisierung versus Absolute Lokalisierung

#### Relative Lokalisierung

- auch: **lokale, inkrementelle Lokalisierung** oder 'tracking'
- Relativ zu einer Startpose wird sukzessiv die Änderung der Pose an diskreten, aufeinanderfolgenden Zeitpunkten ermittelt und integriert

#### Absolute Lokalisierung

- auch als **globale Lokalisierung** bezeichnet

### 3.4 Transformation von Koordinatensystemen lokale $\leftrightarrow$ globale

- Die Pose wird in Bezug auf ein externes Bezugssystem ermittelt, z.B. einer Karte oder einem globalen Koordinatensystem

**Ziel** Bestimme oder schätze die Position und Orientierung des Roboters in seiner Umgebung basieren auf

- der Eigenbewegung
- durch Messungen der relativen Position zu unterscheidbaren Objekten in der Umgebung in Roboterkoordinaten (Ultraschall, Laser, Kamera)

## 3.4 Transformation von Koordinatensystemen lokale $\leftrightarrow$ globale

**Kinematik** Die Kinematik ist die Lehre der Beschreibung von Bewegungen von Punkten im Raum. Dabei werden die Größen Weg, Geschwindigkeit und Beschleunigung betrachtet. Die Kinematik ist ein Teilgebiet der Mechanik.

### Kinematische Robotermodell

- kreisförmiger Roboter
- Zweiradantrieb
- Bewegung in der Ebene

### Lokales Koordinatensystem

- mit dem Roboter verbunden
- Ursprung in der Mitte der Antriebsachse
- x-Achse zeigt in Richtung des Roboterfrontteils

### Festlegung der Roboterposition im globalen Koordinatensystem

- durch die Koordinaten  $M(x_M, y_M)$  im globalen Koordinatensystem
- durch den Winkel  $\theta$  zwischen der lokalen x-Achse und der globalen x-Achse

### 3.5 Karten für statistische und dynamische Umgebungen

- Pose  $p$  gegeben durch:  $p = (x_M, y_M, \theta)^T$

#### Transformation von lokalen in globale Koordinaten

- Koordinaten von  $P$  im lokalen Koordinatensystem:  $p_l = (x_l, y_l)^T$
- Koordinaten von  $P$  im globalen Koordinatensystem:  $p_g = (x_g, y_g)^T$
- Transformation von  $p_l$  nach  $p_g$  ( $m = (x_M, y_M)^T : p_g = R(\theta)p_l$ )

#### Transformation von globalen in lokale Koordinaten

- Transformation von globalen in lokale Koordinaten  $p_l = R(\theta)^{-1}(p_g - m) = R(-\theta)(p_g - m)$

## 3.5 Karten für statistische und dynamische Umgebungen

- Generell gilt: **Karten** sollen eine explizite Repräsentation des Raumes sein.
- Die Karten sind auf die Sensorik des Roboters zugeschnitten
- Die Karten sind nicht vorrangig für den menschlichen Betrachter bestimmt, sondern der Roboter soll sie effizient nutzen können.

#### Statische Umgebungen

- basierend auf der Annahme, dass sich zwar der Zustand des Roboters innerhalb der Umgebung, nicht jedoch die Umgebung selbst ändert.
- Karte spiegelt die wirkliche Umgebung wider.

#### Dynamische Umgebungen

- Objekte können ihre Lage oder ihren Zustand ändern
- In der Karte registrierte Objekte können verschwinden, nicht registrierte Objekte auftauchen
- 'Lernende' Karten sind ein fundamentales Problem in der mobilen Robotik

### 3.5.1 Mapping Methoden

#### Weltzentriert

- Die Pose aller Objekte einschließlich des Roboters werden in der Umgebung in Bezug auf ein festes Koordinatensystem repräsentiert.
- **Indoor**: Ursprung kann eine Zimmerecke sein
- **Outdoor**: Benötigung eines globalen Koordinatensystems, wie z.B. die Längen- oder Breitengrade, i.d.R. nutzen von **WGRS**(World Geographic Reference System)

#### Roboterzentriert

gebraucht um bspw. Kollisionen zu vermeiden

- Durch Koordinaten-Transformation kann zwischen den verschiedenen Referenz-Frames konvertiert werden

### 3.5.2 Arten von Modellen

- Die wichtigste Form von Umgebungsmodellen für mobile Roboter sind Umgebungskarten.
- Die folgende Ausführungen beziehen sich auf geeignet Karten für **mobile, autonome Landfahrzeuge**

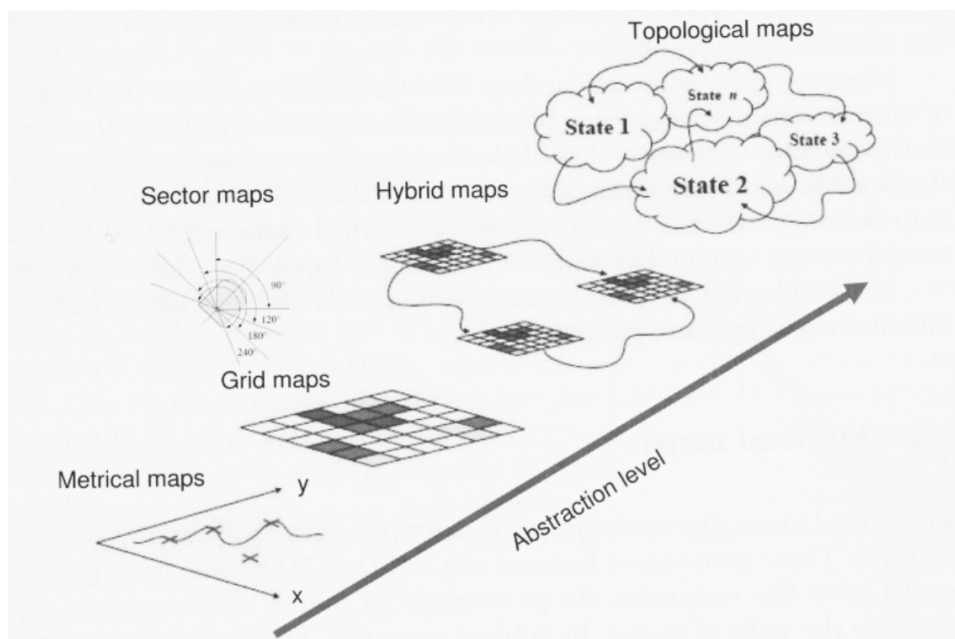


Figure 3.1

#### Arten von Umgebungsmodellen

- **kontinuierliche metrische Karten**, zweidimensional oder dreidimensional:  
jedes Objekt wird assoziiert mit Koordinaten
- **diskrete metrische Karten, Grid Maps**, zweidimensional oder dreidimensional:  
der Raum wird gleichmäßig oder ungleichmäßig aufgeteilt; Objekte werden mit Positionen innerhalb des Gitters assoziiert
- **Hybrid Maps**
- **Topologische Modelle** nur zweidimensional  
im Vordergrund steht die Beziehung der Objekte zueinander

#### 3.5.3 Kontinuierliche Metrische Karten

- Metrische Lokalisierung, beruht auf Ultrashall oder Laserscannern
- Exakte Beschreibung der Umgebung mit 2D oder 3D Modellen möglich
- Die Positionen von Objekten der Umgebung werden durch ein Koordinatensystem repräsentiert.
- **Vorteil:** detailliertes Bild der Umgebung
- **Nachteil:** große, unstrukturierte Datenmengen erschweren die Pfadplanung

#### 3.5.4 Grid Maps - Rasterkarten

- Die Umwelt des Roboters wird in ein gleichmäßiges Raster oder Grid zerlegt.
- Jede Zelle enthält den Belegtheitsgrad der Zelle  $\Rightarrow$  zeigt an ob zelle mit einem Hindernis belegt ist oder nicht
- Verschiedene Modelle verwenden unterschiedliche Werte  
zwei Werten  
freie Zellen, belegte Zellen und Zellen mit Mischbelegung  
Prozentsatz der Belegungswahrscheinlichkeit

### 3.5 Karten für statistische und dynamische Umgebungen

- Notwendige Informationen sind z.B.:
  - x, y als Koordinaten (Zeile, Spalte) einer Zelle
  - Sensordaten des Roboters
  - ein boolescher Wert für den Zustand der Zelle
- Die Werte in den Zellen sind unabhängig voneinander
- Eine Steigerung der Messgenauigkeit der Sensoren führt dazu, dass die Rasterelemente immer kleiner werden
- Für eine kompakte Notation können Grid Maps im 2-dimensionalen Raum mit Quadrees im 3-dimensionalen mit Octrees gespeichert werden

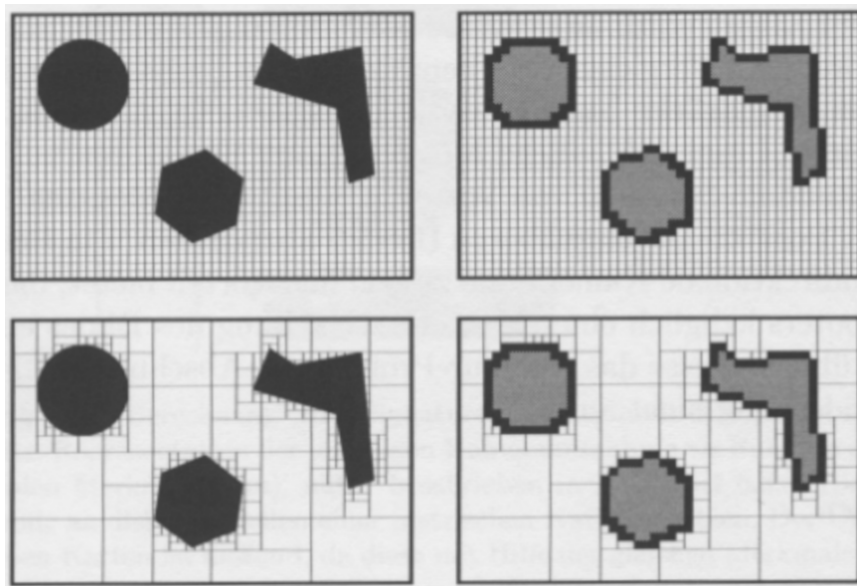


Figure 3.2

#### **Gleichmäßige Gitterstruktur vs. Adaptiver Gitterstruktur**

- links oben zeigt die Objekte in einer gleichmäßigen Gitterstruktur
- rechts oben zeigt die zugehörige Repräsentation über Belegtheiten der Zellen
- speicherintensiv bei gleichmäßiger Unterteilung des Raums  $\Rightarrow$  adaptive Unterteilung des Raums und Speicherung in Quadrees oder bei 3. Dimension Octrees

### 3.5.5 Adaptive Unterteilung

- Ausgangszustand: Rechteck mit Hindernissen
- Fläche wird unterteilt in 4 Rechtecke gleicher Größe
- Jedes Rechteck wird rekursiv wieder in 4 Rechtecke unterteilt  $\Rightarrow$  Quadtree
- Attributierung der Knoten:
  - Frei:** Rechteck enthält keinen Teil eines Hindernisses
  - Belegt:** Rechteck ist vollständig von Hindernis belegt
  - Gemischt:** Rechteck enthält Punkte, die zu einem Hindernis gehören, sowie solche die es nicht tun
- Nur gemischte Knoten werden weiter unterteilt

**Vorteile** schnell und leicht feststellbar, ob Punkt in einem Hindernis liegt

#### Nachteile

- Konturen der Objekte und der Freiraum zwischen ihnen wird unpräzise repräsentiert
- Um die Datenfülle zu reduzieren, wird das Raster zu grob gewählt und dadurch ein möglicher Weg durch Mischpixel versperrt

**Weiterer Verwendungszweck** Neben der reinen Lokalisierung können die Karten auch dazu verwendet werden eine Fahrspur (Trakektorie) zu berechnen.

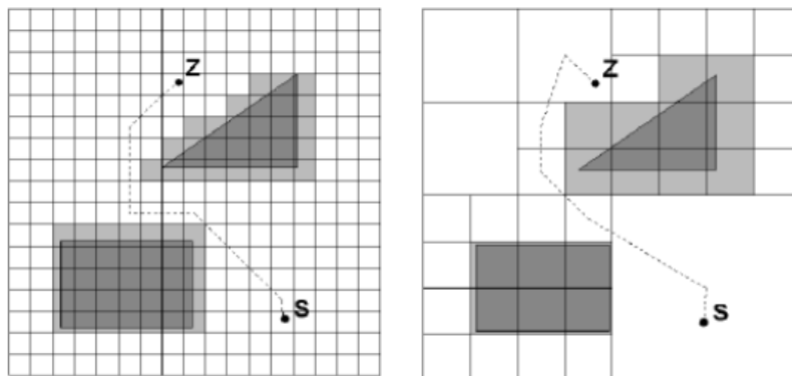


Figure 3.3



### 3.5.6 Weitere Beispiele für Umgebungskarten

- Laserscan Karten
- Bildbasierte Karten

### 3.5.7 Topologische Karten

Bedingt geeignet zur Lokalisation, Haupteinsatzgebiet ist die Pfadplanung.

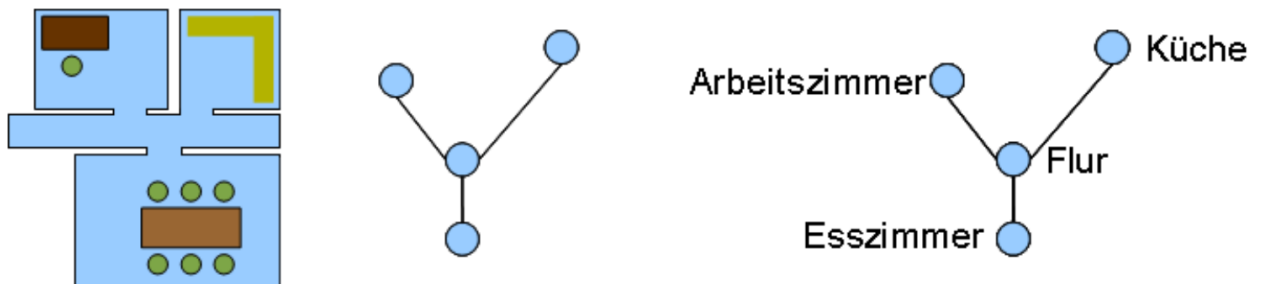


Figure 3.4

- Modelle bilden einen **Graphen**
- **Knoten** entsprechen Orten oder Bereichen der Umgebung
- Beziehungen zwischen den Orten werden durch **Kanten** modelliert.
- Zwei Knoten sind durch eine Kante verbunden, wenn sie unmittelbar voneinander erreichbar sind.
- **Gewichte**: Maß für die Länge der Wege
- Ist die Länge der jeweiligen Wegstücke bekannt, lässt sich der kürzeste Weg finden.

#### Vorteile

- Kompaktheit
- Gute Skalierbarkeit für welträumige Umgebungen.
- Es gibt viele schnelle Algorithmen auf Graphen, die gut zur Pfadplanung eingesetzt werden können

### 3.6 Passive und Aktive Selbstlokalisierung

**Nachteil** Relevante Umgebungsmerkmale werden verdeckt. Landmarken werden schwerer erkannt.

#### 3.5.8 Hybrid Maps

- Kombinieren metrische und topologische Ansätze
- Ermöglichen Lokalisation und Kantenerstellung mit hoher Präzision
- Erhalten die Kompaktheit der topologischen Ansätze

#### Abstraktions-basierter Ansatz

- Basis: konstruieren einer metrischen Karte der Umgebung
- $\Rightarrow$  aufbau einer kompakten topologischen Repräsentation
- **Vorteil** Effizient Planung eines Pfads zu einem gegebenen Ziel aufgrund der Abstraktion.
- Die zugrunde liegende metrische Karte wird für Relokalisation und Hindernisvermeidung benötigt.

## 3.6 Passive und Aktive Selbstlokalisierung

### 3.6.1 Passive Verfahren

- bestimmen oder schätzen die Roboterposition mittels aktueller Sensorinformationen
- beeinflussen **nicht** die Bewegung und Orientierung des Roboters
- Lokalisierungsmodul beobachtet nur die Roboteroperationen
- Roboter bewegt sich zufällig hin und her bzw. führt die zu erledigende Aufgabe durch

### 3.6.2 Aktive Verfahren

- besitzen vollständige oder teilweise Kontrolle über die Bewegungen des Roboters und Ausrichtung der Sensoren
- fährt gezielt bestimmte Orte an um Mehrdeutigkeiten zwischen mehreren Orten aufzulösen

## 3.7 Landmarken

### 3.7.1 Definition

- Als Landmarken werden **eindeutig identifizierbare Charakteristiken der Umwelt** bezeichnet, die von entsprechenden Sensoren erkannt werden können.

#### Landmarke

- ihre Position im Weltmodell ist bekannt
- sichtbar von unterschiedlichen Positionen aus
- erkennbar unter verschiedenen Belichtungen und Blickwinkeln
- relative Position bestimmbar
- stationär, oder dem Navigationsmechanismus muss die Bewegung bekannt sein

**Vorteil** Navigation erfolgt mit der Umwelt selbst und nicht mittels errechneter Daten

### 3.7.2 Natürliche Landmarken

- Werden nicht zum Zweck der Positionsbestimmung aufgestellt, können aber dafür verwendet werden
- Grundsätzlich Passiv

### 3.7.3 Künstliche Landmarken

- Markante Objekte, eigens zum Zweck der Positionsbestimmung in der Umgebung installiert.

## 4 Fortbewegung, Lokalisierungsalgorithmen

### 4.1 Relative Lokalisierung

#### 4.1.1 Dead Reckoning

- **Koppelnavigation oder Dead Reckoning** ursprünglich in der Nautik verwendet
- Mathematisches Verfahren - **Vorwärtskinematik** - zur Positionsbestimmung
- Ausgehend von einer Startposition ist es dem Navigator möglich, seine **aktuelle Position** zu **berechnen** aufgrund der **zurückliegenden bekannten Kurs- und Geschwindigkeitswerte**

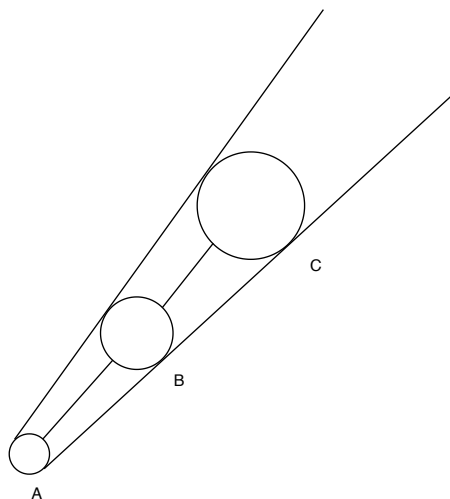


Figure 4.1

- A sei ein gegebener Ausgangspunkt
- Radius wird angegeben der zur Abweichung proportional ist, bsp. hier 0,5m.
- Der Radius spiegelt die mit der Zeit kumulierte Ungenauigkeit wieder

## 4.1 Relative Lokalisierung

- Eine mögliche Roboterposition ist dann innerhalb des Sektors gegeben der durch die Linien eingegrenzt wird

### Vorteile

- einfache Implementierung
- leichte Interpretation der Daten
- unkomplizierte Bedienung
- passable Kurzstreckengenauigkeit

### Nachteile

- Startposition muss bekannt sein
- Genauigkeit nimmt mit zunehmender Länge der befahrenen Strecke drastisch ab

## 4.1.2 Odometrie

Odometrie ist die Wissenschaft der Positionsbestimmung eines Fahrzeugs durch die Beobachtung seiner Räder.

### Grundlegendes verfahren

- Sensoren an Rädern messen Drehbewegung
- **Relative Positionsbestimmung:** Die **Bestimmung der Position** erfolgt ausgehend von einer bekannten Position durch Berechnung des zurückgelegten Weges und anhand von Daten über den Roboter selbst.
- Es wird Inkrementalgebern die Anzahl  $n$  der Radumdrehungen zwischen zwei Messpunkten gezählt. Aus dem bekannten Radumfang wird die Wegdifferenz berechnet mit:

$$\Delta = \Pi \times d \times n$$

- **Ausrichtung** kann durch differentiale Odometrie erfolgen: es werden z.B. die unterschiedlichen Entfernungen gemessen, die die linken und rechten Räder zurückgelegt haben.

## 4.1 Relative Lokalisierung

### Vorteile

- kostengünstig
- hohe Abtastraten
- passable Kurzzeitgenauigkeit

### Fehlerquellen

- Fehlerhafte Messung des Raddurchmessers
- Raddurchmesser nicht gleich, Unrundheit des Radess

### Fehlerberücksichtigung

- Die **Fehler** fließen in die Positionsdiverenz ein, werden zur letzten bekannten Position hinzuaddiert und **summieren sich mit jedem Messschritt**
- Fehlerellipse wächst mit zurückgelegtem Weg
- Odometrie als alleiniges verfahren nur für kurze Strecken geeignet
- Fehler lassen sich bei geringen Geschwindigkeiten und geringer Beschleunigung reduzieren

### 4.1.3 2D-Scanmatching

- Ausgangslage sind zwei Scans, ein Scan M (**Modell**) und ein zweiter Scan D (**Daten**)
- Es wird eine Transformation des einen Scans berechnet und zwar so, dass beide optimal überlagert werden
- Die Transformationen bestehen nur aus einer Rotation und einer Translation
- Die Überlagerung ist optimal, wenn Punkte, die in der realen Szene nahe beieinander liegen, auch in den registrierten Messdaten nahe beieinander liegen.
- **Ziel:** Fehlerfunktion minimieren  $\Rightarrow$  Abstände der Punkte des einen Scans zu ihren korrespondierenden Punkten des zweiten Scans
- Die Transformation des zweiten Scans entspricht dann der Bewegung des Roboters zwischen der Aufnahme der Daten; durch sukzessiven Vergleich kann damit die Bewegung des Roboters nachverfolgt werden

## 4.1 Relative Lokalisierung

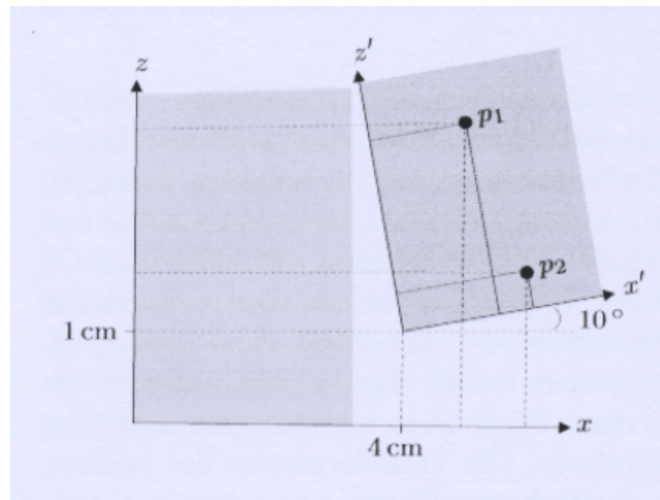


Figure 4.2

### Iteratives Vorgehen

- **Annahme:** die korrespondierenden Punkte sind bekannt  $\Rightarrow$  eine **Transformation** kann berechnet werden, die diese Mengen aufeinander abbildet
- **Beispiel** zwei Scans mit einer Poseänderung des zweiten um  $(4\text{cm}, 1\text{cm}, 10\text{deg})^T$  beide Scans sehen dieselben Raumpunkte  $p_1$  und  $p_2$
- Obige Annahme i.d.r. nicht erfüllt  $\Rightarrow$  **nicht eindeutig zu bestimmen, welche Punkte zwischen den beiden Scans korrespondieren**
- **Lösung: iteratives Vorgehen**, bei dem zunächst eine **Schätzung** der Punktepaarung stattfinden und die Pose des zweiten Scans unter dieser Paarung optimiert wird.
- Iterativ werden mit dem transformierten Scan neue Punktepaare berechnet, bis ein Abbruchkriterium erfüllt ist, d.h. bis sich die Transformation zwischen zwei Schritten nicht mehr signifikant ändert

### Transformationsberechnung

- **Gesucht:** Mögliche Menge von Translationen und Rotationen, unter denen ein korrektes Matching möglich ist.
- $(t_x, t_z, \theta)^T$ , die eine Translation um  $t_x$  in x-Richtung und  $t_z$  entlang der Z-Achse durchführt, sowie eine Rotation um den Winkel  $\theta$
- Der Scan M besteht aus einer Menge von Punkten  $(m_i)_{i=1,2,\dots,N}$

#### 4.1 Relative Lokalisierung

- Der Scan D besteht aus einer Menge von Punkten  $(D_i)_{i=1,2,\dots,N}$

**Minimum der Funktion**  $E(\theta, t) = \sum_{i=1}^N \|p_i - (\mathbf{R}_\theta \mathbf{p}'_i + \mathbf{t})\|^2$

**Transformation zur Minimierung der Fehlerfunktion E** Folgende Transformation mit ggb. Parametern minimiert die Fehlerfunktion:

$$\theta = \arctan\left(\frac{S_{zx'} - S_{xz'}}{S_{xx'} + S_{zz'}}\right)$$

Hierbei ist

$$t_x = c_x - (c'_x \cos \theta - c'_z \sin \theta) \text{ und}$$

$$t_z = c_z - (c'_x \sin \theta + c'_z \cos \theta)$$



# 5 Navigation

## 5.1 Bekanntes vs. unbekanntes Terrain

Man unterscheidet zwischen Algorithmen für:

- **bekannte Umgebungen** (auch während der Fahrt ändert sich die Umgebung nicht)
- **unbekannt Umgebung**
- Gebiet **vollständig bekannt**  $\Rightarrow$  Lösung der Suche mittels eines Graphen.
- **unvollständig oder gar nicht bekanntes Gelände**  $\Rightarrow$  Berechnungen erfolgen auf **lokalen Teilinformationen**

Neue Informationen  $\Rightarrow$  **inkrementelle Anpassung** über Sensoren

## 5.2 Navigation in unbekanntem Terrain

### 5.2.1 Konturverfolgung

- Eine **Freiraumfahrt** d.h. eine Fahrt durch ein Gelände, dessen Raum möglichst weit und frei von Hindernissen ist  $\Rightarrow$  nicht immer Zielführend
- Lösung: **Konturverfolgung**

Roboter wird nah an einem Objekt (Wand, Hinderniss) entlang bewegt

Es sollte möglichst ein gegebener Abstand  $d$  eingehalten werden

### Regelung des Abstands $d$

```
1 IF (Distance to Wall > d)
2   THEN turn to wall
3
4 IF (Distance to Wall < d)
5   THEN turn away from wall
6
7 IF (Distance to Wall == d)
8   THEN Drive straight ahead
```

### 5.2.2 Sensorbasierte Planer - Navigation mit Hinderniskontakt

**Bug1 Algorithmus** Fordert zwei bestimmte Verhalten:

- bewege dich auf einer geraden Linie
- folge einer Begrenzung in einem bestimmten Abstand

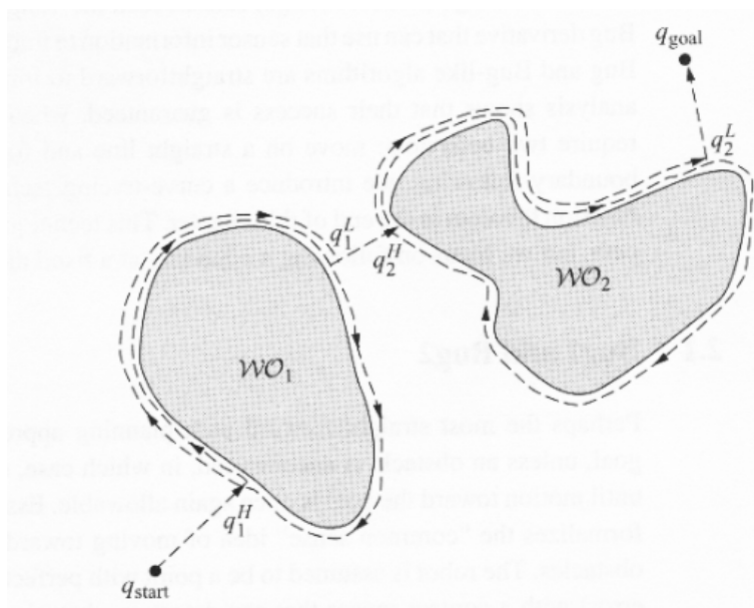


Figure 5.1: Bug1 Algorithmus Route

### Vorraussetungen

- Roboter benötigt einen Sensor zur Erkennung eines 'Kontakts' mit einem Hindernis

## 5.2 Navigation in unbekanntem Terrain

- Roboter kann die Distanz zwischen zwei Punkten  $x$  und  $y$  messen
- der Arbeitsraum ist begrenzt

### Algorithmus

- Roboter folgt Linie zum Ziel bis er beim Punkt  $q_1^H$  auf ein Hindernis trifft
- Roboter umfährt das Hindernis bis er erneut beim gleichen Punkt auf das Hindernis trifft
- Roboter bestimmt den nächsten Punkt (Leavepoint)  $q_1^L$  vom Hindernis zum Ziel und fährt diesen Punkt an
- Von diesem Punkt fährt der Roboter geradewegs zum Ziel bis er das Ziel erreicht oder erneut auf ein Hindernis trifft

**Exception** Schneidet die Linie von  $q_1^L$  zum Ziel das **akutelle Hindernis** gibt es keine Lösung

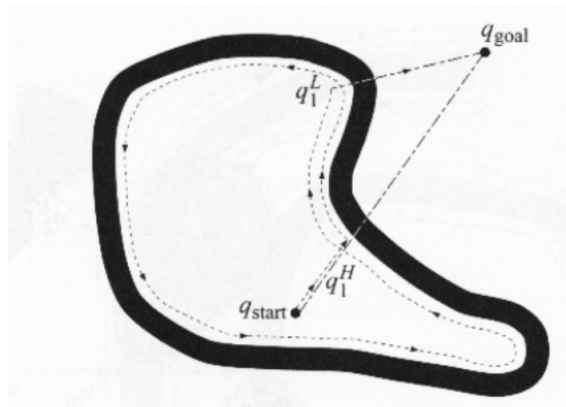


Figure 5.2

### Bug2-Algorithmus Idee:

- Linie vom Start  $S$  zum Ziel  $T$  konstruieren
- bewegt sich möglichst auf dieser Linie
- Roboter trifft auf Objekt  $\Rightarrow$  umfahren in einer bestimmten Richtung bsp. immer rechts herum

## 5.2 Navigation in unbekanntem Terrain

- Hindernis wird solange umfahren, bis der Roboter wieder auf einen Punkt auf der Linie  $ST$  trifft, der näher zum Ziel liegt als der ursprüngliche Kontaktpunkt mit dem Hindernis

### Algorithmus:

---

```
1 1. Auf Geraden  $ST$  in Richtung  $T$  fahren bis:
2   a.) Ziel erreicht wird --> END
3   b.) Hindernis getroffen wird --> Schnittpunkt  $H_j$  setzen
4       --> Schritt 2
5 2. Umfahre das Objekt bis:
6   a.) das Ziel erreicht wird --> END
7   b.) Gerade  $ST$  in einem Punkt  $Q$  getroffen wird mit Strecke  $QT$  kleiner als
      Strecke  $H_jT$ 
8        $QT$  wird in Richtung  $T$  verlassen
9        $Q$  als Leavepoint  $L_j$  setzen
10       $i$  um 1 erhöhen
11      --> Schritt 1
12 c.) zum letzten Hitpoint  $H_j$  zurückgekehrt wird
13     Algorithmus wird abgebrochen
14     Es gibt keinen Weg zu  $T$ 
```

---

- Ev. wird kein Weg zum Ziel gefunden
- Bei Komplexen Hindernissen kann Ziel mit **Backtracking** gefunden werden

## 5.2 Navigation in unbekanntem Terrain

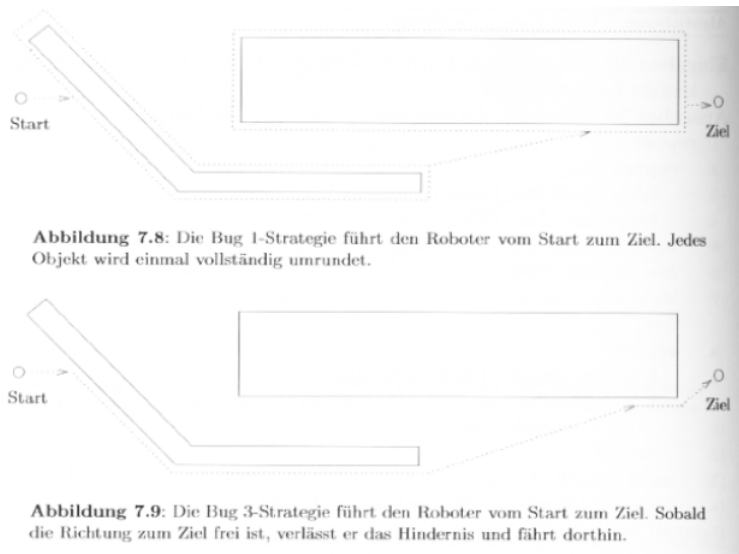


Figure 5.3

### Bug3-Algorithmus

#### 5.2.3 Labyrinth

##### Grundprobleme

- Es wird davon ausgegangen, dass der Roboter berühren oder 'sehen' kann
- Zwei grundlegend Hauptprobleme:

Einen **Weg in ein Labyrinth** finden, um einen bestimmten Gegenstand oder **Schatz zu erreichen** sowie den **Rückweg zum Eingang**

**Flucht aus einem Labyrinth** von einer unbekannten Stelle aus.

- Enger Zusammenhang zwischen Labyrinth und Graphen  $\Rightarrow$  Jeder Korridor:= Kante und jede Kreuzung:= Knoten

Bei bekanntem Labyrinth  $\Rightarrow$  Suchproblem auf in Bäumen

**Verlassen eines Labyrinths mit Pledge Algorithmus** Grundidee: Vorsichtig geradeaus bis man auf ein Hindernis trifft und dann mit der 'linken' Hand immer an der Wand entlang bis zum Ausgang.

**Problem:** Enthält das Labyrinth eine Säule, läuft man für immer im Kreis

## 5.2 Navigation in unbekanntem Terrain

- **Lösung**  $\Rightarrow$  man folgt der Wand nur solange, bis man wieder in die alte Richtung schaut.

**Allgemeingültige Lösung:** Drehungen beim Abbiegen an den Ecken mitzählen. Bei jeder Linksdrehung wird der Umdrehungszähler inkrementiert, bei jeder Rechtsdrehung dekrementiert.

- Bewege den Roboter geradeaus bis eine Wand erreicht ist
- Folge der Wand bis Umdrehungszähler 0 ist

### Verlassen eines Labyrinths mit Ariadnefaden

- **Ziel:** einen Weg zu einem versteckten Ziel im Labyrinth sowie wieder zurück zum Eingang finden ohne dass eine Karte des Labyrinths bekannt ist
- **Idee:** Wenn man ein Labyrinth betritt Faden ausrollen  $\Rightarrow$  zurückverfolgen bringt einen zurück zum Eingang.

### Vorraussetzungen und grundsätzliches Vorgehen

- einer Wand folgen
- Umdrehen
- Kreuzungen erkennen
- Ziel erkennen
- Faden auslegen und wieder einsammeln
- Faden am Boden erkennen
- Faden zur nächsten Kreuzung folgen

### Tarry und Tremaux Algorithmus

- Beispiel für klassische Tiefensuche
- Richtung, in der sich das zu suchende Objekt befindet ist unbekannt.
- Graph kann zyklen enthalten
- Es wird ein zyklisch gerichteter Graph durch jede Kante konstruiert, wobei jede Kante nur einmal pro Richtung besucht wird

### 5.3 Pfadplanung für mobile Roboter in bekanntem Terrain

#### Algorithmus:

- Starte willkürlich an einem Knoten
- Folge einem möglichen Pfad, markiere die Kante, in welcher Richtung sie betreten worden ist
- Sind alle Kanten schon betreten, eine auswählen, die bis jetzt nur in die Gegenrichtung betreten wurde.
- Trifft man auf eine Sackgasse oder einen schon besuchten Gang, zurück zur letzten Kreuzung
- Es darf kein Pfad betreten werden, der schon in beide Richtungen besucht wurde.
- Algorithmus ist beendet, wenn der Startpunkt erreicht wird.

## 5.3 Pfadplanung für mobile Roboter in bekanntem Terrain

### 5.3.1 Bewegungsplan für mobile Roboter

Ziel der Navigation ist es, ein Fahrzeug in der Umwelt zu bewegen. Dies beinhaltet drei Unteraufgaben:

#### Globale Pfadplanung

- **Vorraussetzung:** es gibt eine Karte
- suche eines Pfades von einem Start- zu einem Zielpunkt in vorhandenem Umgebungsmodell
- evtl. auch Suche nach Pfad mit geringsten Kosten
- Kompletter Pfad beschrieben durch Menge von Punkten

**Lokale Pfadplanung** berücksichtigt Fahrzeug-Dimension und kinematische Einschränkungen

**Path Control** generiert geeignet Steuerbefehle, um den vorberechneten Pfad zu folgen

### 5.3.2 Konfigurationsraum

#### Herleitung

- Abmessungen, Form, Bewegungsmöglichkeiten des Roboters werden für die Erstellung des Konfigurationsraums benötigt
- Konfiguration  $q$  eines Roboters beschreibt Lage und Ausrichtung im Bezugssystem des Umgebungsmodells
- Im zweidimensionalen Raum kann Position in  $x, y$ -Ebene und Orientierung ausgedrückt werden
- Konstruktionsbedingt sind einige Konfigurationen für den Roboter in seiner Umgebung nicht zulässig
- Problem; einfachere Darstellung:

Roboter als Punkt angenommen

Abmessungen des Roboters + Objektanmessungen  $\Rightarrow$  Konfigurations-Hindernisse

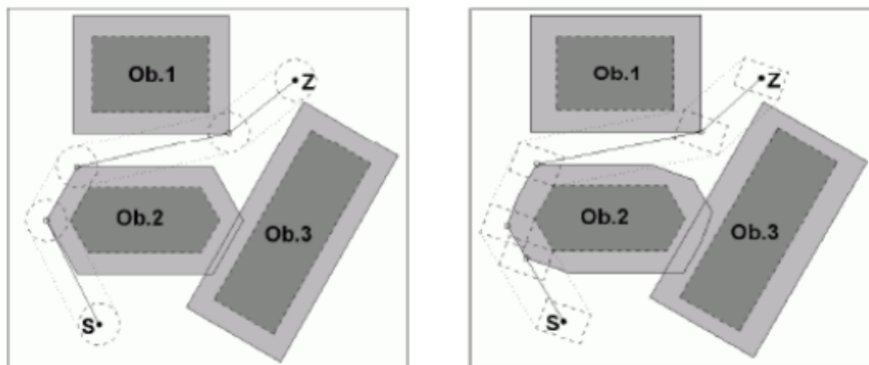


Figure 5.4

#### Definition

- Summe aller Konfigurations-Hindernisse bildet den **Konfigurationsraum**
- Konfigurationsraum ist Datenstruktur, die es dem Roboter ermöglicht, die Position und Orientierung von Hindernissen in der Umgebung zu definieren
- Der Konfigurationsraum dient als Basis der Wegplanung



### Repräsentationen

- Graphen mit Knoten
- Reguläre Gitter
- Quad-Bäume oder Octal-Bäume oder als Voronoidiagramme

## 5.4 Algorithmen und Methoden

Für die folgenden Algorithmen und Methoden wird ausgegangen, dass Hindernisse bekannt sind und weder Position noch Form ändern

**Zellerlegungen** das Umgebungsmodell wird in sich nicht überlappende Zellen unterteilt, die als besetzt oder frei markiert sind

### Roadmaps

- Das entstehende Netzwerk muss **topologisch** alle zwischen den Hindernissen befahrbare Wege umfassen.
- Planer kann dann kollisionsfreien Pfad von Start- zu Zielpunkt erstellen

Auf dieses Netzwerk können Standardmethoden der Graphentheorie, wie sie auch in der Autonavigation Verwendung finden, angewendet werden:

- kürzeste Wegsuche mit A\*, Dijkstra
- Wegsuche mit Umgehung von Hindernissen mit dem **Sichtbarkeitsgraph**
- Wegsuche mittels eines **Voronoigraphen, Voronoidiagramms**

**Potentialfeldmethoden** beinhalten die physikalische Simulation des Roboters als Partikel in einem Feld.

### 5.4.1 Dijkstra

#### Datenstrukturen

- **PriorityQueue** *Open*
- **Aktueller Knoten**  $n$
- **Nachfolgerknoten**  $n'$
- **Vorgänger** *predecessor*
- **Startknoten**  $s$
- **Kosten** *cost*

#### Funktionsweise

### 5.4.2 A\*

#### Datenstrukturen

- **PriorityQueue** *Open*
- **List** *Closed*
- **Aktueller Knoten**  $n$
- **Nachfolgerknoten**  $n'$
- **Vorgänger** *predecessor*
- **Startknoten**  $s$
- **Kosten**  $g = \text{bekannteKosten}$ ,  $h = \text{Schätzkosten}$ ,  $f = \text{Gesamtkosten}$

#### Funktionsweise

### 5.4.3 Wegsuche und Umgehung bekannter Hindernisse mit dem Sichtgraph-Algorithmus

#### Visibility Map

## 5.4 Algorithmen und Methoden

- enthält Eckpunkte von Polygonen, den Ecken der Hindernisse
- zwei Knoten der Visibility Map teilen eine Kante, wenn die beiden Eckpunkte voneinander in Sichtweite sind

### Visibility Graph

- ist die einfachste Visibility Map
- Knoten umfassen: **Startknoten**, **Zielknoten** und alle Eckpunkte der Hindernisse
- die Kanten sind Liniensegmente, die zwei Knoten in Sichtweite verbinden
- Hinderniskanten sind Teil des Sichtgraphen

### Sichtgraphenalgorithmus

- verbinde **Start**-, **Zielpunkte** und die **Ecken** der Hindernisse durch Geraden, die nicht durch Hindernisse laufen dürfen
- Ergebnis ist ein Graph, dessen Knoten Orte sind und dessen Kanten mögliche Wegstücke zwischen diesen Knoten darstellen
- Kanten sind gewichtet mit der Entfernung zwischen den Knoten
- gesucht: Menge der möglichen Geraden, die den Startpunkt auf dem kürzesten Weg mit dem Zielpunkt verbindet
- **Nachteil**: führt unmittelbar an Hindernissen entlang, enthält noch viele nutzlose Kanten

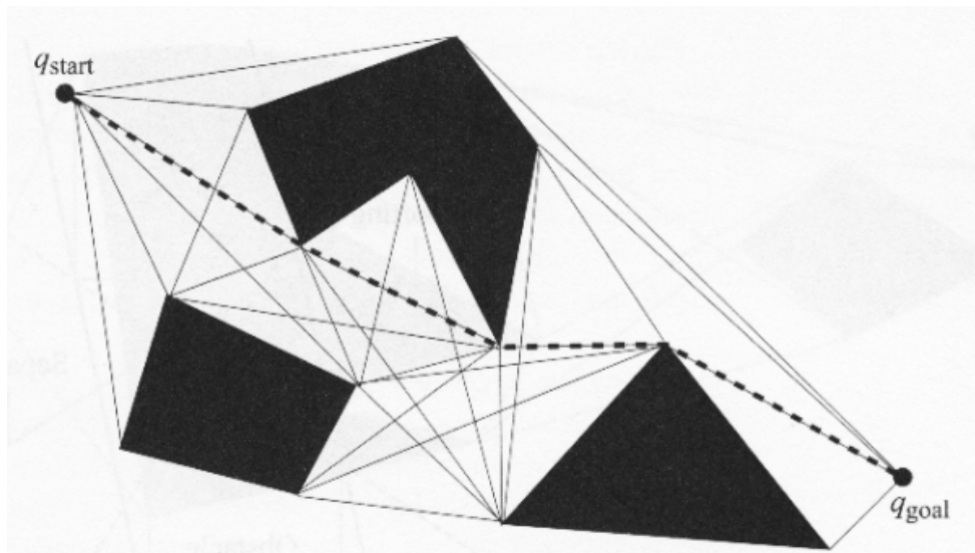


Figure 5.5

**Reduzierter Visibility Graph** Definition von **unterstützenden** und **trennenden** Kanten:

- **unterstützende Kante:** Tangente zu zwei Hindernissen, so dass die Hindernisse auf derselben Seite der Linie liegen
- **trennende Kante:** Tangente zu zwei Hindernissen, so dass die Hindernisse auf gegenüberliegenden Seiten der Tangente liegen

Der reduzierte Sichtgraph besteht nur aus unterstützenden und trennenden Kanten.

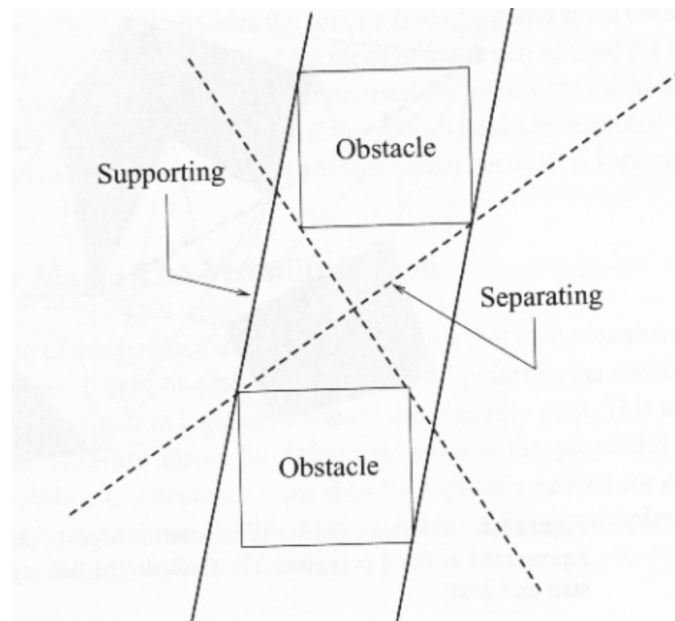


Figure 5.6

### Konstruktion des Graphen

- Alle Liniensegmente  $vv_i$  mit  $v! = v_i$  müssen getestet werden, ob sie keinen Schnittpunkt mit einer Kante eines Hindernisses haben
- **Nachteil:** Komplexität  $O(n^3)$
- **Effizienter:** Plane Sweep Algorithmus mit Komplexität  $O(n^2 \log n)$

### Algorithmus

- **Input:** A set of vertices  $v_i$  (whose edges do not intersect) and a vertex  $v$
- **Output:** A subset of vertices from  $v_i$  that are within line of sight of  $v$
- For each vertex  $v_i$ , calculate  $\alpha$ , the angle from the horizontal axis to the line segment  $\overline{vv_i}$
- Create the vertex list  $\epsilon$ , containing the  $\alpha_i$ 's sorted in increasing order.
- Create the active list  $\mathcal{S}$ , containing the sorted list of edges that intersect the horizontal half-line emanating from  $v$ .
- **For all  $\alpha_i$  do**
  - if  $v_i$  is visible to  $v$  **then**
    - Add the edge  $(v, v_i)$  to the visibility graph.
  - end if**
  - if  $v_i$  is the beginning of an edge,  $E$ , not in  $\mathcal{S}$  then**
    - Insert the  $E$  into  $\mathcal{S}$
  - end if**
  - if  $v_i$  is the end of an edge in  $\mathcal{S}$  then**
    - Delete the edge from  $\mathcal{S}$
  - end if**
- **end for**

### 5.4.4 Voronoi-Diagramme

#### Allgemeines Voronoi-Diagramm

- Eine Fläche wird willkürlich mit Punkten besetzt  $\Rightarrow$  erzeugt Polygonflächen
- Alle punkte einer Polygonfläche liegen am nächsten zu dem Punkt der im Zentrum dieses Polygons liegt
- Jede Zelle hat genau ein Zentrum

## 5.4 Algorithmen und Methoden

- Die Punkte des Gebiets, die zu mehreren Zentren den gleichen Abstand haben, bilden die Grenze zwischen den einzelnen Zellen
- Als Abstandsfunktion wird der Euklidische Abstand verwendet:

$$\text{dist}(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

- Das Voronoi-Diagramm ist die Menge solcher Grenzlinien

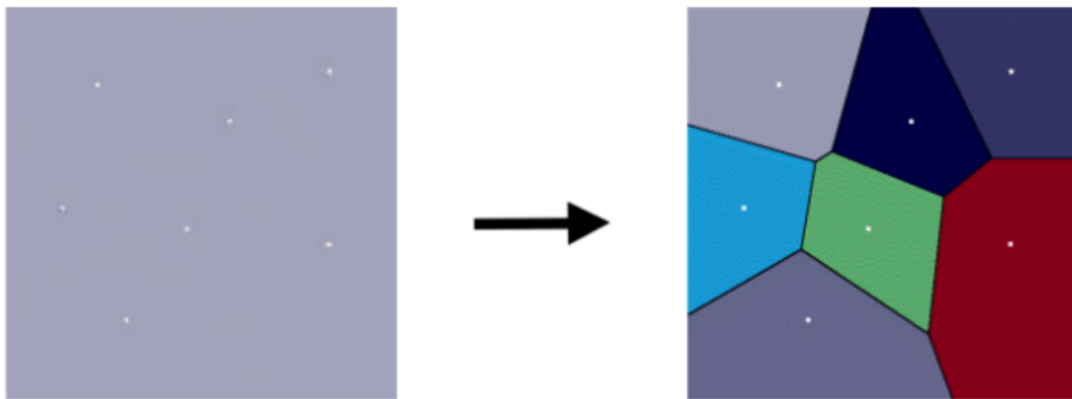


Figure 5.7

Immer dann gut einsetzbar, wenn ungenaue Sensoren zur Verfügung stehen oder die Umwelt ungenau geometrisch modelliert wurde oder sich dynamisch ändert.

### Generalisierte Voronoi-Diagramme

- Bei der Navigation von Robotern wird eine verallgemeinerte Version des Voronoi Diagramms verwendet, das sogenannte **generalisierte Voronoi Diagramm (GVD)**
- **Generalisierung** betrifft die **Form der Zentren** und die Art der **Abstandsfunktion**
- **Zentren** können aus **komplexeren Formen** wie Linien, Kurven oder Polygonen **statt Punkten** bestehen
- Objekten der Umgebung werden als **Voronoi-Zentren** behandelt
- Die Menge aller **Voronoi-Kanten**, das GVD, stellt mögliche **kollisionsfreie Wegstücke** dar
- Falls sich der Roboter entlang einer **Voronoi-Kante** bewegt, kann er nicht mit Hindernissen kollidieren

## 5.4 Algorithmen und Methoden

- Beliebtes Verfahren zur Repräsentation des Konfigurationsraumes und Erzeugung eines Graphen.
- Punkte an denen sich **Voronoi-Kanten** schneiden, werden zu **Voronoi-Knoten**

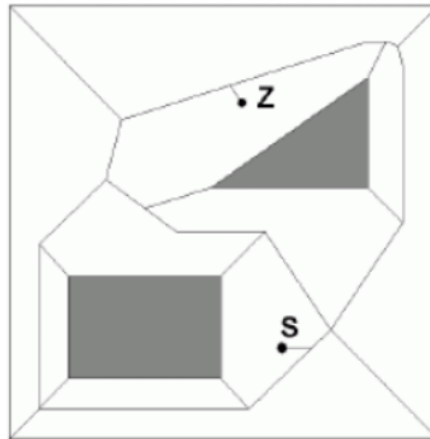


Figure 5.8

Start- und Zielpunkt des gesuchten Weges liegen normalerweise nicht auf dem Diagramm. Diese beiden Punkte werden mit der nächstliegenden Kante verbunden. Die Berechnung des optimalen Weges auf dem Voronoi-Diagramm kann mit üblichen Graphenalgorithmien vorgenommen werden.

### 5.4.5 Navigation in einer Rasterkarte

- Die Karte ist durch ein binäres Raster mit freiem Platz und Hindernissen gegeben.
- Das Raster wird ausgehend vom Startpunkt: **geflutet**  $\Rightarrow$  Füllt den gesamten hindernisfreien Raum
- In jeder Iteration haben alle Pixel auf einer Wellenfront dieselbe Pfadlänge im Raster bezogen auf den Zielpunkt
- Backtracking vom Zielpunkt zurück zum Startpunkt und erstellt dabei eine Liste der passierten Rasterpunkte
- Gewählt kann jeder Punkt im Raster werden, dessen Wert eins geringer ist, als der aktuell betrachtete
- Trifft dies auf mehrere Punkte zu, wird willkürlich einer gewählt



#### 5.4 Algorithmen und Methoden

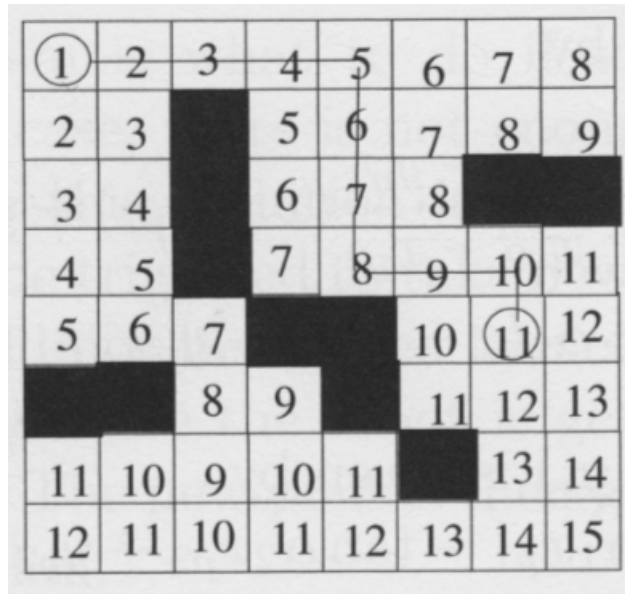


Figure 5.9

#### Fluten der Karte

- Ein **Rasterpunkt** besitzt vier Nachbarpunkte: 'N', 'O', 'S', 'W'
- Ein Rasterpunkt  $z$  ist mit 0 für freien Raum belegt und -1 für ein Hindernis
- Wenn ein Rasterpunkt eine Nummer erhalten hat, behält er diese Nummer
- Der Algorithmus benutzt aus Performancegründen zwei Stacks  $S_0$  und  $S_1$ , die abwechselnd gefüllt und geleert werden. Die Stacks sind anfangs leer

---

**Algorithm 7.7** Flooding a raster with thick walls

---

```

Initialization:  $i := 1; j := 0;$ 
starting cell  $z := 1$ ; push  $S_i$ ;
repeat
   $i := (i + 1) \bmod 2; j := (j + 1) \bmod 2$ 
  repeat
    pop stack  $S_i$ ; check surrounding of the cell taken out of stack:
    for all cells with  $z = 0$  do
      if  $\min_{z>0}(O,S,W,N) > 0$  then
         $z := \min_{z>0}(O,S,W,N) + 1$ ; push to stack  $S_j$ ;
      else
        do nothing;
      end if
    end for
  until stack  $S_i$  is empty
until stack  $S_j$  is empty
  
```

---

Figure 5.10

**Alternativer Wave Front Planer** Siehe Ursprungsfolie Kapitel 5, Seite 29/30

### 5.4.6 Potentialfeldmethode

#### Grundidee

- Vorbild: **Elektrisches Feld**
- Start- und Zielpositionen, die Positionen aller Hindernisse müssen bekannt sein.
- Der **Zielpunkt und Freiräume** erhalten ein **anziehendes Potential**
- Der **Startpunkt, die Hindernisse** und Wände erhalten ein **abstoßendes Potential**
- Es wird eine Karte generiert mit virtuell anziehenden und abstoßenden Kräften
- Kräfte nehmen linear mit dem Abstand zu den Objekten ab.
- Ein Objekt bewegt sich nach der Methode des steilsten Abstiegs im Potentialfeld auf das Ziel zu.

## 5.4 Algorithmen und Methoden

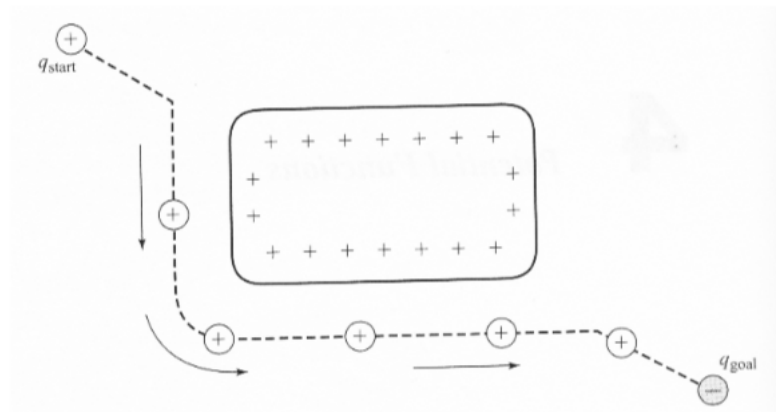


Figure 5.11

# List of Figures

2.1	Schema der Hybridmodell Schichten . . . . .	7
2.2	Filesystem, das ROS zugrunde liegt . . . . .	10
3.1	. . . . .	17
3.2	. . . . .	19
3.3	. . . . .	20
3.4	. . . . .	21
4.1	. . . . .	24
4.2	. . . . .	27
5.1	Bug1 Algorithmus Route . . . . .	30
5.2	. . . . .	31
5.3	. . . . .	33
5.4	. . . . .	36
5.5	. . . . .	40
5.6	. . . . .	41
5.7	. . . . .	43
5.8	. . . . .	44
5.9	. . . . .	45
5.10	. . . . .	46
5.11	. . . . .	47

# Listings