Robotik

Verena Hamburger
verena@joschs-robotics.de

Vorbemerkung

Die Robotik ist wohl eines der interessantesten und fantasieumwobensten Anwendungsgebiete der Künstlichen Intelligenz. So war der Golem von Prag einer der vielen rein mythologischen Robotern, und obwohl das Wort „Roboter“ (vom slawischen robota für arbeiten) einer Science-Fiction-Erzählung von Karel Capek („Rossum's Universal Robots“, 1921) entsprang, wurde der Begriff „Robotik“ von SF-Autors Isaak Asimov geprägt. In seinem Buch „Runaround“ (1942) legte er drei fundamentale Regeln der Robotik fest, deren oberstes Gebot besagt, daß ein Roboter keinem Menschen Schaden zufügen darf.
Die Herausforderung in der Robotik besteht nicht nur in der Wahl einer adäquaten Hardware, sondern auch in der Erstellung der passenden Software, die das Planen einer Problemlösung und das Auftreten von unvorhersehbaren Problemen einschließen muß. Dies ist der Ansatzpunkt für die Methoden der KI in der Robotik.

Inhalt

1. Einleitung

Definition eines Roboters:
Ein Roboter ist ein aktiver, künstlicher Agent, der in der realen Welt agiert.
Im folgenden sprechen wir hauptsächlich von einem autonomen, mobilen Serviceroboter, der ihm gestellte Aufgaben selbständig löst. Mobile Roboter werden dabei kurz als **Mobot** bezeichnet.

Die reale Welt - sei sie teilweise bekannt oder a priori unbekannt- stellt dabei hohe Anforderungen an den Mobot, denn sie ist für ihn:

- ***Nicht direkt zugänglich***, da der Mobot nur über externe Sensoren Zugang zu seiner näheren
- Umgebung hat, wobei auch Meßfehler und -toleranzen einkalkuliert werden müssen.
- ***Nicht deterministisch***, d.h. die Zukunft bzw. die Auswirkungen einer geplanten Aktion können nicht vorhergesagt werden. Unvorhergesehene Hindernisse, Durchdrehen von Rädern, Stromschwankungen etc. können die Ausführung eines Plans verhindern.
- ***Nicht episodenhaft***, d.h. es ist nicht gewährleistet, daß eine Aktion unter gleichen Umständen immer gleich abläuft bzw. zum selben Ergebnis führt. Deswegen muß der Mobot fähig sein, interaktive Entscheidungen zu treffen, um einen Plan neu anzupassen. Er muß wissen, welche Ereignisse es wert sind, sich daran zu erinnern d.h. zu lernen.
- ***Dynamisch***. Da sich die Welt ständig im Wandel befindet, muß der Mobot entscheiden, ob er eine Aktion sofort ausführen muß, oder ob er sie auf einen späteren Zeitpunkt verschieben kann, z.B. kann ein freier Durchgang plötzlich durch ein anderes sich bewegendes Objekt versperrt werden.
- ***Kontinuierlich***. Im Gegensatz zur simulierten Computerumgebung ist die reale Welt nicht aus diskreten Zuständen zusammengesetzt, sondern veränderlich in ganz $\Re$. Es kann also nicht jede Bewegung, die der Mobot ausführen muß vorhergesagt werden, weshalb auch keine klassischen Such- und Planalgorithmen verwendet werden können..

2. Anwendungsbereiche für Roboter

Traditionell werden Roboter in der Fließfertigung der Automobilindustrie eingesetzt. Diese bieten sich besonders zur Automatisierung an, da die Tätigkeit immer exakt dieselbe ist. So patentierte George Devol 1954 den programmierbaren Roboterarm. Ein Beispiel für einen frühen Roboterarm ist der Stanford Manipulator dargestellt in Abb. 1. Doch da

die Umgebung statisch ist und die Arbeit dieser Roboter weder Mobilität und Sensoren noch Verfahren der künstlichen Intelligenz erfordert, soll nicht weiter darauf eingegangen werden.

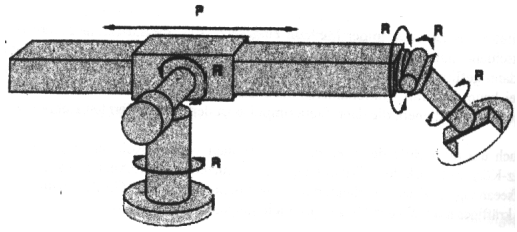


Abb.1: Stanford Manipulator mit 5 rotierenden und einem prismatischen Gelenk

Der nächste Schritt ist die Entwicklung teleoperierender Systeme (sog. Telepräsenz), die von einem menschlichen Benutzer geleitet bzw. ferngesteuert, lebensgefährliche Arbeiten, wie z.B. das Warten von Brennstäben in einem Atomkraftwerk, erledigen. Solche Systeme wurden nicht nur bei den Aufräumarbeiten nach dem Tschernobyldisaster eingesetzt, sondern sie helfen New Yorker Polizisten jeden Tag beim Entschärfen von Bomben. Problematisch wird dieses Konzept, wenn wie bei der Pathfindermission auf dem Mars, die Befehlslaufzeit zu lange wird (30 min einfach). Der zweite Nachteil teleoperierender Systeme liegt in der Tatsache, daß sie während der gesamten Arbeitszeit ständig von einem Menschen kontrolliert sein müssen. Ihre „Intelligenz“ liegt dabei in der Feinabstimmung der Bewegungen.
Ihre Nachfolger, autonom gelenkte Fahrzeuge, können sich in einer preparierten Umgebung selbständig bewegen, da sie durch Sensoren einen Weg, der z.B. durch Induktionsschleifen oder andere Landmarken vorgegeben ist, als Wegweiser benutzen. (Auf das benötigte Verfahren zur Wegplanung werde ich später genauer eingehen.) Sie werden hauptsächlich im Materialtransport eingesetzt. Der Nachteil dieser zwar autonomen Fahrzeuge liegt klar an den Eingriffen in die Umwelt und der Eingeschränktheit des Roboters auf diese speziell angepasste Umgebung.

Trotzdem hat ein Roboter im Allgemeinen viele Vorteile:

- hohe Verfügbarkeit, da er 24 Stunden ohne Pause arbeiten kann.
- hohe Zuverlässigkeit. Ein Roboter vergißt oder verliert nichts, und hat außerdem keine persönlichen Beweggründe bzw. Ziele.
- Jeder Arbeitsschritt kann nachvollzogen und überwacht werden.
- höhere Genauigkeit als beim Menschen (wichtig in der Mikroelektronik)
- Risikominderung für Menschen, wie z.B. bei den teleoperierenden Systemen.

Heutzutage gehen die Bemühungen mehr in die Richtung des mobilen Serviceroboters (siehe Kapitel 1), der selbständig aufgrund von Sensordaten Entscheidungen trifft, Hindernisse umgeht und in der uns geläufigen, d.h. nicht speziell vorbereiteten, Umgebung Dienstleistungen erbringt. Solche Mobots putzen, liefern auf Anfrage Post und mehr, überwachen Gebäude, werden als Blindenhund eingesetzt u.v.m.

3. Bestandteile eines Roboters

Im nächsten Abschnitt sollen kurz die wichtigsten Bestandteile und Grundbegriffen der benötigten Hardware zur Realisierung von Robotern erklärt werden.

3.1 Effektoren

Grundsätzlich besteht jeder Roboter aus starren **Gliedmaßen** (links), die paarweise durch **Gelenke** (joints) verbunden sind. Man unterscheidet Roboter anhand ihrer Ausstattung, d.h. Sensoren und Effektoren. Durch **Effektoren** beeinflußt der Roboter seine Umwelt (Manipulation). Um ihn zu kontrollieren bzw. zu bewegen (Lokomotion), sind sie mit **Aktuatoren** bestückt, die Software in physikalisch Bewegungen umwandeln. Spezielle Effektoren, die am Ende einer Gliedmaße mit der Umwelt interagieren, heißen **Endeffektoren**. Diese können Schraubenzieher, Sprühdosen, Schweißgeräte, etc. sein.

3.1.1 Lokomotion

Es gibt viele Möglichkeiten für einen Mobot sich fortzubewegen. Dabei sind anatomische Beine zwar elegant, aber gleichermaßen ineffizient und schwer zu realisieren. Sie werden hauptsächlich in sehr unebenen Gelände eingesetzt, da sie stabilen Halt garantieren. Ketten eignen sich ebenfalls gut zum Überwinden unebener Stellen, haben jedoch einen geringen Wirkungsgrad. Am häufigsten werden daher Räder verwendet, da diese leicht zu kontrollieren sind und schnelle Fortbewegung auf glatten Flächen gewährleisten.
Zusammenfassend kann man sagen: je komplexer der Antrieb, desto einfacher die Bewegungsplanung und umgekehrt. Deshalb ist es sehr wichtig je nach Aufgabengebiet die richtige Balance zu finden.
Man unterscheidet zweite Kategorien von Robotern *holonomic* und *nonholonomic*, wobei die zweite im Gegensatz zur ersten weniger Steuergrade als tatsächliche Freiheitsgrade hat (z.B. einen minimalen Wendekreis statt auf dem Punkt zu drehen).

Im dreidimensionalen Raum, also auch in der realen Welt, stehen uns sechs Freiheitsgrade zur Verfügung (siehe Abb.2).

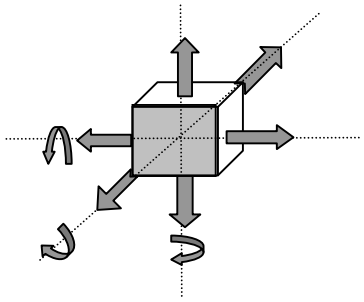


Abb. 2: drei rotatorische und drei prismatische Freiheitsgrade im dreidimensionalen Raum

3.1.2 Manipulation

Effektoren, die Objekte bewegen, nennt man **Manipulatoren**, deren Vorfahren teleoperierende Systeme sind (vgl. Kapitel 2).
Um die Form oder die Eigenschaften eines Objektes zu verändern, muß man zuerst den benötigten Endeffektor an die vorgesehene Position bewegen. Jedes Gelenk fügt dem Roboter einen zusätzlichen **Freiheitsgrad** hinzu. Früher hat man oft versucht die Hand des Menschen nachzubilden. Der Vorteil des menschlichen Arms liegt in einem redundanten Freiheitsgrad, der es unseren Fingern erlaubt, am Platz zu bleiben, während unser Arm einem Hindernis ausweicht. Jedes Gelenk erlaubt entweder **Rotations- oder Translation-(Schub-)Bewegung**, die durch primitive elektrische Signale- Ein/Aus oder skaliert- realisiert wird. Bald stellte man fest, daß die Nachbildung einer ganzen Hand sehr komplex, jedoch der Einsatz eines Zwei- bis Drei-Finger-Greifers zuverlässig und einfach zu kontrollieren ist. Das wichtigste hierbei ist das Umsetzen eines Tast- und Kraftsinnes wie im folgenden näher erläutert.

3.2 Sensoren

Sensoren unterteilt man in drei Hauptgruppen:

- Interne Sensoren messen die inneren Zustände des Roboters.
- Externe Sensoren erfassen die Eigenschaften und Zustände seiner Umwelt.
- Oberflächensensoren messen Kontakteigenschaften mit anderen Objekten

Um hier nur die wichtigsten zu nennen:

- Der **proprioseptische Sinn**, d.h. die Wahrnehmung der Eigenbewegung (Odometrie) im Bezug auf den umliegenden Raum, wird durch Enkoder realisiert, die die Stellung der Gelenke auslesen. Dieser wichtige Sinn ist allerdings mit großer Unsicherheit behaftet, da ein Rad z.B. durchdrehen kann oder durch einen Einstellungsfehler die Richtung falsch angenommen wird. Hinzu kommt die Meßungenauigkeit der Sensoren. Trotzdem liefert dieser Sinn gute Orientierung- und Geschwindigkeitswerte.
- Der **Kraftsinn** (force sense), angebracht zwischen Manipulator und Endeffektor, mißt die Kraft d.h. Druck und Torsion, die der Roboter in allen Freiheitsgraden auf ein Objekt wirken läßt. Dadurch ist es ihm möglich, an einer Wand entlang zu gehen, während er mit einem vorgegebenen Druck Kontakt hält.
- Der **Tastsinn** (touch- oder tactile sense) mißt die Verformung und Vibrationen eines berührten Objekts. Durch ein Feld von Meßpunkten auf der elastischen Oberfläche des Endeffektors ist es dem Mobot möglich, ein Ei von einem Buch zu unterscheiden und beides so zu halten, daß es weder zerbricht noch herunterfällt.
- **Kameras** sind wohl die komplizierteste Art, ein Umweltmodell zu erstellen, da aus den gelieferten Bildern die relevanten Informationen extrahiert und ihnen eine Bedeutung (Semantik) zugeordnet werden muß. Dies kann durch Landmarken, wie Barcodes, erleichtert werden.
- **Sonar** (Ultraschall mit ca. 50 kHz) wird häufig zum schnellen Ausweichen von Hindernissen und zum Erstellen einer Karte eingesetzt, da dieser Sinn die Distanz zum nahesten Hindernis mißt. Problematisch wird diese Methode bei unebenen Oberflächen, die den Schall in alle Richtungen reflektieren (Streuung). So können durch Schallwellen, die über Umwege zum Sensor zurück reflektiert werden, oder durch zusätzliche Schallquellen sog. „Geister“ auftreten, d.h. Hindernisse, die gar nicht vorhanden sind. Deswegen müssen Sonardaten anhand von Wahrscheinlichkeitsrechnung vorbehandelt werden.
- **Laser** wird prinzipiell genau wie Sonar eingesetzt, hat aber den Vorteil der kürzeren Wellenlänge und des schmaleren Ausbreitungskegels. Beide werden als Tiefensensoren verwendet und erstellen eine dreidimensionale Karte der Umgebung. Abb. 3 zeigt links die grundsätzliche Funktionsweise eines Lasersensors. Und rechts, wie der Agent seine Umgebung als Entfernungsbild wahrnimmt. So erkennt er nicht, daß ihn eine Stange den Weg versperrt, sondern lediglich, daß ein rundes längliches Objekt aus der restlichen Umgebung hervorsteht. Weiterhin hat ein Laser den Vorteil, daß er mehr Bilder pro Zeit erkennt, d.h. Veränderungen werden schneller wahrgenommen.

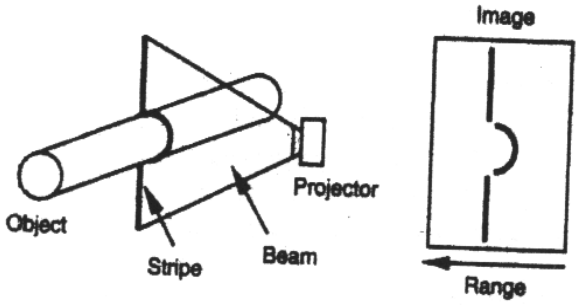


Abb.3: Wahrnehmung eines Objekts mittels vertikaler Lichtstreifen

Sensoren liefern meist ihre Werte zur besseren Kontrolle und eventuellen Korrekturen an den verantwortlichen Aktuator zurück (siehe Abb. 3). Dabei beeinflussen nicht nur die Sensordaten, sondern auch der jeweilige Zustand des Mobots die Einstellung der Aktuatoren. Moderne Roboter verwenden meist mehrere Sensoren gleichzeitig, die manchmal sogar zwischen widersprüchlichen Daten abwägen müssen.

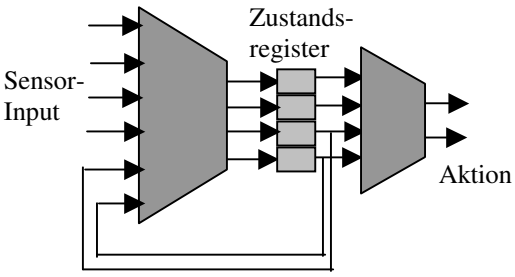


Abb.4: Grundsätzlicher Aufbau von Rosenscheins Situated Automata

4. Roboterarchitekturen

Die Architekturen bestimmen wie ein Roboter seine Aufgaben löst und wie er den riesigen Sensorinput bewältigt. Es kann allerdings keine generelle Wertung vorgenommen werden, die eine der Architekturen als besser bevorzugt, da hierfür keinerlei Beweis existiert.

4.1 Klassische Architektur aus den späten 60ern

Der erste mobile Roboter war Shakey (siehe Abb. 5). Er konnte sich auf ebenem Boden fortbewegen und dabei große Objekte schieben. Ausgerüstet mit einer Kamera, versuchte er, einfache geometrischer Figuren zu erkennen, was allerdings z.B. bei Seilen nicht immer möglich ist. Desweiteren nutzte er einen zweidimensionalen Wegplanungsalgorithmus und einen Resolutionsbeweiser, der leider zu ineffizient war.

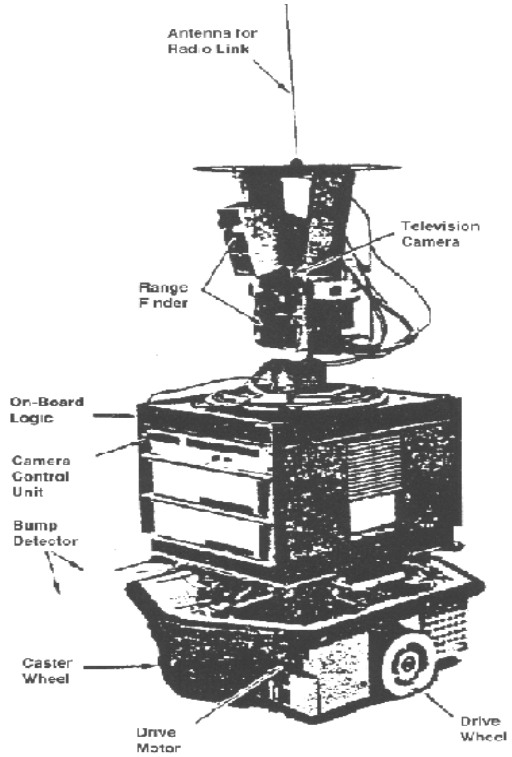


Abb. 5: Shakey (1969)

Ein weiteres, großes Problem war es, das Fehlschlagen von Plänen einzukalkulieren, weshalb selten einer von Shakeys Plänen zum Erfolg führte. Es wurde allerdings bei Shakeys zweiter Version, die auch sonst viele Verbesserungen hatte, explizit behandelt.
Sein Planungsystem benutzte den **Strips-Algorithmus**, der das Wegfindungs- und Manipulationsproblem in spezielle Teilzielprogramme, die sog. **ILA**s **(Intermediate-Level Action)**, zerlegt. Jedes ILA bestand dabei wiederum aus einer Routine verschiedener primitiver **LLA**s, die als sog. **Low-Level Aktionen** den physikalischen Roboter kontrollierten und die inneren Zustände aktualisierten. Hinzu kam noch etwas Fehlererkennung und –behebung. Wahrscheinlichkeiten für Bewegungsfehler wurden bei jedem Schritt berücksichtigt und sobald die Unsicherheit zu groß war, d.h. einen Schwellwert überstieg, wurde ein passendes Unterprogramm zu erneuten Positionsbestimmung aufgerufen. Der neue Shakey hatte nun eine ziemlich verläßliche Basis, um seine Aufträge auszuführen, wobei die ILA/LLA Struktur die einzige nicht klassische Komponente ist.
Ausgeführt und überwacht wurden die so erstellten Pläne von **Plantex**, das gleichzeitig für die ständige Überprüfung des aktuellen Weltmodells zuständug war. Indem es die Vorbedingungen für jeden ferti-

gen Plan mit seinen Sensordaten verglich, fand es die entsprechende Lösung. Paßte keine der Vorbedingungen zur aktuellen Situation oder schlugen die „Instant"-lösungen fehl, entwarf Strips einen neuen Plan. Dadurch wurde Shakey auch mit unvorhergesehenen Problemen fertig.
Seine Rechenzeit betrug dabei ca. 15 min für jeden Schritt. So wurde in der verbesserten Version von Shakey erstmalig die Idee aufbrachte, erfolgreiche Planungsergebnisse in Makros zu speichern. So konnte Strips durch „Erinnern" an eine gleiche oder ähnliche Situation die Planungszeit um ein Vielfaches reduzieren. Dieses Lernen aufgrund der Speicherung und Erklärungen ausgewählter Beispiele nennt man **Explanation Based Learning (EBL)**. Später mußte man allerdings feststellen, daß explizite Beweisführung der Effekte von Low-Level-Aktionen für Echtzeit Verhalten zu aufwendig ist.

4.2 Situated Automata

Ziel der Situated Automata ist es daher, explizite Überlegungen durch endliche Zustände zu ersetzen, wodurch eine effektive Implementierung von **reagierenden Zustandsagenten** (reflex agents with state). Hierzu gibt es zwei verschiedene Ansätze:

1. Stan Rosenschein unterschied dabei zwei Arbeitsschritte (1985):

- ***Große Strategie:*** Offline Kompilierung des vom menschlichen Programmierer explizit präsentierten Wissens
- ***Große Taktik:*** explizites Wissen innerhalb der Roboterarchitektur

Aus den inneren Zuständen und den physikalischen Gesetzen lassen sich bestimmte logische Annahmen und Merkmale über die Umwelt ableiten. Daher werden diese Informationen ständig durch eine Feed-Forward-Schleife zur Aktualisierung der Zustände verwendet (siehe Abb. 4). Die verwendete Vorgehensweise nennt man **funktionale oder vertikale Dekomposition** (eng.: Zerlegung, Aufspaltung), wobei das teuerste bzw. aufwendigste die Repräsentation der Umweltzustände ist (Schema siehe Abb.6). Hierbei werden die gesammelten Umweltdaten zu einem Zustand komprimiert und daraus die nächste Aktion und die zugehörigen Aktuatorbefehle abgeleitet (siehe Abb. 4 & 6).

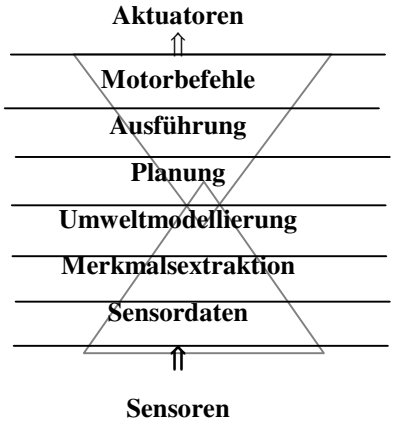


Abb.6: funktionale Dekomposition

2. Rodney Brooks vertrat 1986 die verhaltensbasierte Robotik, die in vielem an die LLA und ILA Struktur von Shakey erinnert, jedoch dessen bloßes Reagieren auf Umstände viel weniger Zeit benötigt als beim errechneten Vorgehen durch LLAs. Im Gegensatz zu Rosenschein, setzte er auf die verhaltensgesteuerte oder horizontale Dekomposition. So unterscheidet der Roboter verschiedene Verhaltensweisen oder Reflexe, die er beherrschen muß. Der Agent vertraut also nicht auf ein zentrales Umweltmodell, sondern hier hat jede Verhaltensebene Zugang auf alle Sensordaten und extrahiert die für den jeweiligen Reflex wichtigen bzw. auslösenden Daten. Hierbei kann ein Verhalten einer höheren Priorität stets auf den inneren Zustand eines niedereren Verhaltens zugreifen und ihn ggf. ändern (siehe Abb.7). Daher werden nur wenige Bits für die inneren Zustände benötigt, was den Aufwand enorm verringert. Dies ist natürlich sehr vorteilhaft für viele Basisqualifikationen, jedoch benötigt man einen neuen Kontroller für jede Aufgabe. Ist der Kompiler in der Robotersoftware enthalten, kann der Agent seinen eigenen Situated Automata implementieren und ist so auf alle möglichen Situationen vorbereitet. Der Automat wird jedoch in einer komplexen Umwelt auf Grund von Rekursion zu groß und eignet sich daher nur für einfache Umgebungen.

Abb. 7: Verhaltensbasierte Dekomposition

5. Konfigurationsraum eines Mobots

Als Konfigurationsraum eines Agenten bezeichnet man die Menge aller Zustände sowohl seiner Umwelt als auch der Körper des Mobots selbst. Da all diese Zustände reellwertig sind, und so gibt es, trotz der starren Gelenke, eine unendliche Zahl von Zuständen und Aktionen. Dies ist der Grund warum keine Standardsuchalgorithmen verwendet werden können.
Nimmt man einen Roboter mit k Freiheitsgraden (pro Gelenk ein Freiheitsgrad), kann man die Position eines Endeffektors als Punkt im k-dimensionalen **Konfigurationsraum C** (vom Englischen configuration space) interpretieren.
Auf seinem Weg durch C muß der Mobot Hindernissen, wie z.B. Tischen und Wänden, ausweichen. Den durch Hindernisse versperrten Raum, der natürlich eine Untermenge von C ist, nennt man dabei **Hindernisraum O** (vom Englischen configuration space obstacles). Dieser schließt auch die Grenzen von C, sowie die (physikalischen) Grenzen des Roboters ein, z.B., daß Gelenke sich nur bis zu einem gewissen Grad verdrehen oder kippen lassen bevor wichtige Kabel (Strom oder Daten) beschädigt werden.
Im restlichen, **freien Raum F** (vom Englischen free space) von C kann sich der Mobot frei bewegen. Das Problem hat sich so auf ein mathematisches Problem reduziert, einen Lösungsweg zu finden, der vollständig in F liegt (sog. **Policy-Funktion π**).
Für Menschen ist dies viel einfacher da sie die Schulter als redundanten Freiheitsgrad nutzen. So ist es uns möglich den Arm zu bewegen, während unsere Hand am selben Ort bleibt.

Der sog. **generalisierte (oder verallgemeinerte) Konfigurationsraum W** enthält zusätzlich zu C die Objektkonfigurationen ε, d.h. alle Zustände von beweglichen oder veränderlichen Objekten im Raum. ε (m-dimensional) kann ebenfalls als Modell zur Beschreibung von nicht geometrischen (und somit für den Roboter schwer beschreibbare) Gegenständen (z.B. ein Seil) verwendet werden.

Problematischerweise lassen sich im k+m dimensionalen W nur seine eigenen k Freiheitsgrade vom Agenten kontrollieren, wenn er mit einem Objekt nicht direkt in Verbindung steht. Es gibt für den Mobot also drei Möglichkeiten: entweder er trägt ein Objekt (**transport path**), er verändert ein Objekt oder er bewegt sich frei im Raum (**transit path**). Konfigurationen, in denen Gegenstände frei in der Luft schweben, sind illegal und werden somit eliminiert. Sind alle Gegenstände außer dem Mobot unveränderlich, gilt W = C. Der steuerbare Teil von W heißt **Foliation**.

Um ungültige Konfigurationen zu vermeiden gibt es drei Möglichkeiten:

1. **Abstraktion**: W wird in endlich viele Zustände unterteilt. Dabei wird der exakte Standort der Blocks ignoriert.
2. **Klassisch**: Der Agent plant zuerst die Objektbewegungen in festen Gruppen und dann seinen eigenen Weg. Diese Vorgehensweise führt zu diskreten Aktionen.
3. **LMT-Ansatz** (benannt nach den drei Autoren): limitiert Objektbewegungen indem er nur einen festen Satz von Basisbewegungen zuläßt. Sind die Restriktionen zu gering wird das Lösen schwierig, sind sie jedoch zu streng wird das Problem unlösbar. Wie bei der Lokomotion (Kapitel 3.1.1) muß auch hier das richtige Maß gefunden werden. Dieser Ansatz behandelt explizit das Unsicherheit des Antriebs und der Sensoren mittels einer Wahrscheinlichkeitswolke, die mit der Zeit wächst (vgl. Shakey).

In dieser Wahrscheinlichkeitswolke, die als Menge von möglichen Konfigurationen (Untermenge von W) beschrieben werden kann, werden meist jedoch die relativ guten, internen Sensoren als perfekt angenommen, d.h. ihre Daten haben die Wahrscheinlichkeit eins und gelten somit als gesichert. Eine solche Menge nennt man **Recognizable Set $\sigma^{-1}(s)$**. Hiebei bezeichnet s die aktuellen Sensorendaten und der abstrakte Sensor σ eine Relation $W \rightarrow \sigma^{-1} \subset W$, die jedem Weltzustand einen Satz möglicher Sensorendaten zuweist. Die verschiedenen Sätze sind disjunkt, aber ihre Vereinigung bildet W. Oft versucht der Planer die nächsten Sensorendaten vorauszuahnen $\sigma^{-1}(s_n) \rightarrow \sigma^{-1}(s_{n+1})$. Manchmal werden auch Erinnerungen (vgl. Shakey) als virtuelle Sensoren verwendet. Leider ist der Aufwand hierfür extrem hoch.

6. Navigation und Bewegungsplanung

Dieses Problem stellt die Hauptsache in der Robotik, da die vorherigen Kapitel lediglich als Werkzeug zu seiner Lösung beitragen. Das wichtigste ist dabei die Frage: „Wo bin ich?“ , da der Mobot zwar gute Kenntnisse über seinen eigenen, nicht aber über den Zustand der Umwelt, hat. Hierbei müssen all in Kapitel eins beschriebenen Umweltprobleme, sowie Meßfehler der Sensoren, nicht erkennbare Hindernisse und Bewegungsfehler behandelt werden..

6.1 Zerlegung in Zellen (cell dekomposition)

Die Zellenzerlegung unterteilt F in geometrisch einfach Zellen, deren Nachbarbeziehungen mittels Adjazenzmatrix dargestellt werden. Als erstes muß der Agent nun die Start und Zielzelle lokalisieren. Dann legt er seinen Weg jeweils durch die Mitte des Zellenrands und durch den Zellenmittelpunkt der zwischen Start und Ziel liegenden Zellen fest (Abb.8). Vorteilhafterweise verläuft dieser Weg immer durch die Zellenmitte, d.h. durch die von den Hindernissen am weitesten entfernten Punkt.
Problematisch wird dies erst in einer engen Umgebung. Da die Grenzen von F nur seltenst den einfachen Formen der Zellen entspricht und wir die Gefahr eines Zusammenpralls verhindern wollen, bleiben bei jeder Zelle sog. **wasted wedges**, das sind Stücke von F, die fälschlicherweise als Hindernis betrachtet werden (siehe Abb.8). Somit findet der Mobot evtl. keinen Weg zum Ziel, obwohl einer bei Nutzung eben dieser kleinen Stücke möglich wäre.

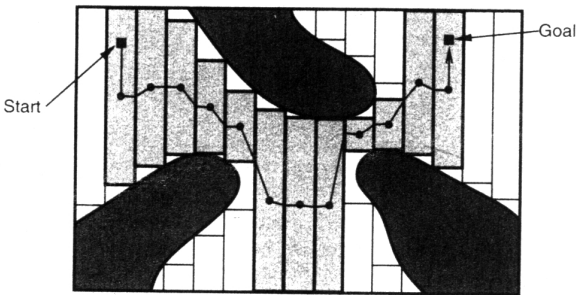


Abb.8: Beispiel zur Zellenzerlegung

Die einfachste Lösung hierfür wäre ein Algorithmus, der an kritischen (schmalen) Stellen, die Zellenbreite verringert, um so die verlorenen Stücke möglichst gering zu halten. (Natürlich ist dies nur bis zu einem gewissen Grad sinnvoll, da der Agent aufgrund von Ungenauigkeiten einen bestimmte Sicherheitsabstand braucht, der eine unterste Grenze für die Zellenbreite bildet.)
Um F möglichst vollständig auszufüllen, wird oft die **zylindrisch Zellenzerlegung** eingesetzt, die auch komplexe Enden der Zellen zuläßt. Dabei muß man zuerst die kritischen Punkte der Hindernisse herausfinden. Dies sind die Punkte, die eine vertikale Tangente haben. Die Zellenbreite definiert sich je zwischen diesen vertikalen Tangenten und der Weg wird je durch die Mittelpunkt dieser Zellen gewählt.
Beide Ansätze setzten jedoch vollständige Kenntnis der Umwelt voraus.

6.2 Navigation auf Skeletten (skeletonization method)

Diese Methode ist einfacher und zeitsparender als die Zellenzerlegung, da sie eine explizite Beschreibung der Begrenzung von F vermeidet und somit eine minimale Beschreibung von F erreicht. Sie rduziert das Wegfindungsproblem auf ein einfaches Graphsuchproblem auf einer eindimensionalen Untermenge von F.
Hierzu gibt es wiederum drei Ansätze*, aber auch hier muss die gesamte Umgebung als bekannt vorausgesetzt werden:*

1. **Sichbarkeitsgraph (visibility graph)**: Man legt den Start- und Zielpunkt des Roboters fest und zieht nun, wie in Abb.9 gezeigt, eine Linie zwischen diesen Punkten und allen sichtbaren Ecken der Hindernisse (d.h. eine solche Linie darf an keinen Punkt im Hindernis liegen). Der Agent muß jetzt nur noch den kürzesten Weg vom Start zum Ziel finden entlang dieser Linien. Man kann diese Methode noch weiter verfeinern, indem man virtuelle Hindernisse kreiert, die die reelen Hindernisse um einen Sicherheitsabstand erweitern, und den Roboter zu einem Punkt reduziert.
Sichbarkeitsgraphen sind v.a. bei größtenteils geometrischen Hindernissen sinnvoll. Der Nachteil ist, daß dieser Weg, selbst wenn es eine direkt Verbindung gibt, immer entlang der Hindernisse verläuft.

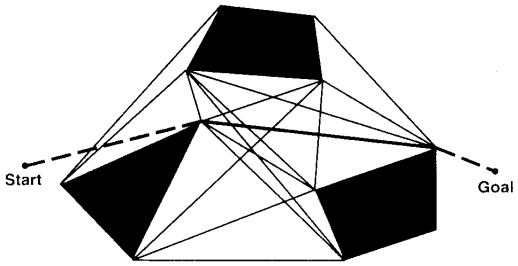


Abb.9: Sichtbarkeitsgraph mehrerer polygonaler Hindernisse

2. **Vornoidiagram**: Hierbei nimmt man an jedem Punkt in F den Abstand zum nähesten Hindernis auf. Man kann es auch vergleichen mit der sog. Potentialfeldmethode. Dieser Ansatz ordnet dem Zielpunkt eine hohen Anziehungskraft zu, wogegen Hindernisse den Roboter abstoßen. Die Abstoßungskraft der Hindernisse nicht mit sinkendem Abstand zu und ist direkt am bzw. im Hindernis unendlich groß. Der Mobot muß nun nur noch den resultieren Kraftvektoren bis zum Ziel folgen (siehe Abb.10).
Der wesentliche Nachteil hierbei ist, daß lokale Minima in einer Sackgasse entstehen können, in die der Roboter gezogen wird und gefangen bleibt.

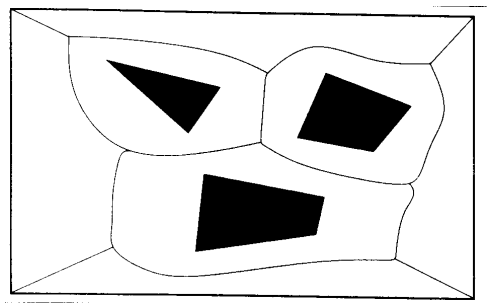


Abb.10: Voronoidiagramm

3. **Straßenkarten (roadmaps)**: Die Benutzung von Straßenkarten ist die effizienteste, vollständige Methode. Sie basiert auf der Idee, die möglichen Wege des Mobots von einer Fläche unendlich vieler Punkte auf wenige, festgelegt „Straßen“ zu reduzieren. Dazu benötigt man zwei Typen von Skeletten: **Silhouetten (oder freeways) und Verbindungen (oder Brücken).** Die Silhouettenmethode beschreibt dabei die sichtbaren Grenzen der Hindernisse indem sie diese als Kurve zwischen zwei Koordinatenachsen X und Y festlegt, und die lokalen Extrema dieser Kurve in konstant breiten Streifen von X berechnet. Aus diesen Maxima und Minima ergibt sich nun die Silhouette. Brücken verbinden dabei kritische Punkte der Silhouette. Diese Methode ist ungünstig, sowohl vom Effizienz-, als auch von Sicherheitsaspekt her, da auch hier die Kurve direkt an den Hindernissen vorbeiführt. Deshalb definiert man die Silhouette meist, ähnlich wie im Vornoidiagramm, als Kurve durch Extremas der Entfernung von Hindernissen (siehe Abb.10 & 11). Brücken führen nun von einem kritischen Punkt direkt zum nähesten Punkt der Silhouette (siehe Abb.11).

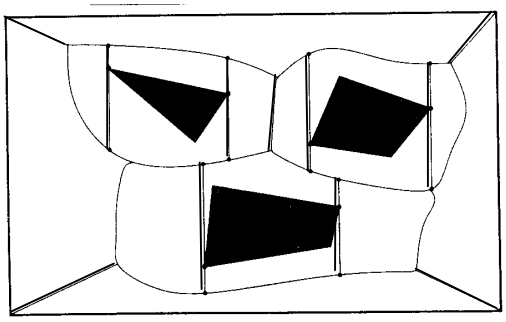
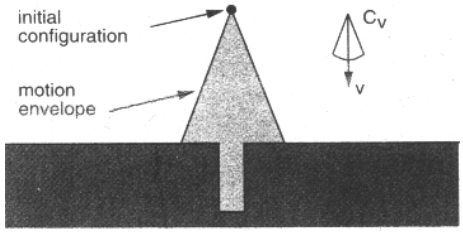


Abb. 11: Voronoi-ähnliche Roadmap

6.3 Planung feiner Bewegungen (fine motion planning)

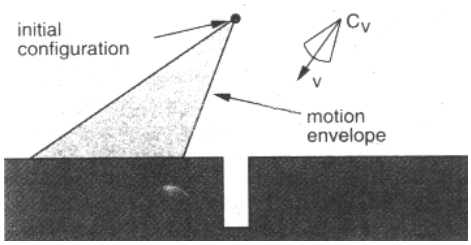
Das hauptsächliche Anliegen dieses Ansatzes ist der Umgang mit Unsicherheiten. Er plant kleine, genaue und überwachte Bewegungen, weshalb der Hauptteil der Arbeit offline ist, d.h. ohne externe Sensoren. Diese Bewegungen setzen sich aus einem Kommando (konstante Geschwindigkeit) und einer Abbruchbedingung zusammen und unterstützen somit ein dynamische Modell zu Kollisionsvermeidung. Dabei sieht man den mit Unsicherheiten behafteten Weg nicht als Gerade, sondern als **Wahrscheinlichkeitskegel** (siehe Abb. 12).
Als Bewegungsmodell wird oft das **Federmodell** verwendet (**spring model**), daß den Weg für eine imaginäre Feder plant, deren eines Ende am Mobot befestigt ist. Hierbei ist die Reaktionsstärke proportional zur relativen Entfernung ihrer Endpunkte. Ähnlich wird auch das **Dämpfermodell** (**damper model**) verwendet. Der Dämpfer, der anstelle der Feder am Mobot „angebracht“ wird, reagiert proportional zur relativen Geschwindigkeit. Beide Modelle erlauben dem Roboter an einer Oberfläche entlang zu gleiten, solange er nicht mit dem Hindernis kollidiert würde.
Eine denkbare Aufgabe für einen Serviceroboter wäre beispielsweise einen Nagel in ein Loch in der



Wand zu stecken (siehe Abb. 12).

Abb. 12: Ausgangssituation um Loch an der (hier schwarzen) Wand zu finden

Betrachtet man nun den Bewegungskegel, ist es ziemlich wahrscheinlich, daß der Mobot das Loch verfehlt. Deshalb wählt man meist einen Weg, der den Agenten absichtlich am Rand des Kegels entlang, d.h. auf jeden Fall am Loch vorbei, führt



(Abb. 13) und läßt ihn nun solange an der Wand entlang gleiten, bis er das Loch findet (Abb. 14).

Abb. 13: Der Agent verfehlt das Loch absichtlich.

Der Agent verfügt über Teilkenntnisse über seine Umwelt und ergänzt den Rest mit Hilfe seiner Sensoren.

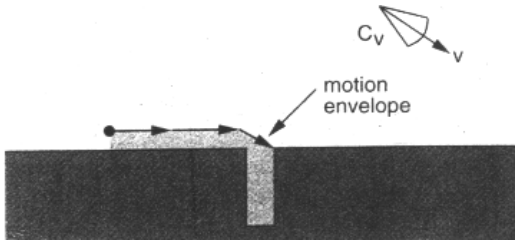


Abb. 14: Der Agent geht auf das Loch zu und ertastet es.

6.4 Navigieren anhand von Markierungen (landmark based)

Das durch Landmarken gestützte Navigieren orientiert sich an leicht erkennbaren Markierungen, z.B. Barcodes, anhand derer der Mobot seine exakte Position bestimmen kann. Da sich der Agent im Einflußbereich bzw. in Sichtweite dieser Landmarken befinden muß, werden diese einheitlich, an strategisch wichtigen Positionen angebracht. Startet der Roboter außerhalb des Einflußbereich einer Markierung, muß eine Startregion angenommen werden. Nehmen wir an, der Zielpunkt des Roboters liegt im Einfluß einer Landmarke, so kann man rückwärts vom Zielpunkt einen Weg zur Startregion planen, der durch den Einflußbereich aller zu passierenden Landmarken führt. Wie auch beim Fine Motion Planning wird die Bewegung als Wahrscheinlichkeitskegel angenommen. Man wählt diesen Kegel so, daß der Mobot auch im schlechtesten Fall im Einflußbereich der nächsten Markierung landet (siehe Abb.15). Von dort aus kann er den Randpunkt aufsuchen, der am nähesten am nächsten Einflußbereich liegt. Diese Vorgehensweise nennt man **Rückwärtsprojektion (Backprojection)**.
Bei dieser Methode hat der Agent vollständige Kenntnis über die Umgebung, jedoch nicht über seine Umwelt.

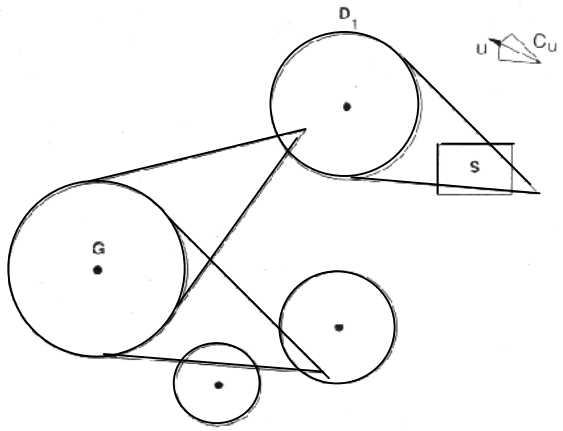


Abb. 15: Backprojection

6.5 Online-Algorithmus

Ist die Umgebung nur wenig oder gar nicht bekannt, ist es unmöglich einen Plan zu erstellen, der unter allen Umständen kollisionsfrei zum Ziel führt. Zwar versucht der Agent eine Policyfunktion zu erstellen, trifft seine Entscheidungen jedoch während der Laufzeit (also online). Die Echtzeit Tauglichkeit und Einfachheit dieses Algorithmus ist daher extrem wichtig, weshalb es hierbei auch keine „Erinnerungen“, d.h. keine Speicherung erfolgreich ausgeführter Pläne und Aufträge. Der Algorithmus lautet wie folgt:

1. Ziehe eine gerade Linie zwischen Start- und Zielpunkt.
2. Gehe solange auf dieser Linie bis ein Hindernis den Weg versperrt und merke dir deinen Standpunkt.
3. Umkreise das Hindernis bis du wieder am Ausgangspunkt ankommst. Speichere dabei alle Überquerungen der Linie.
4. Gehe nun den kürzesten Weg zurück zu dem Schnittpunkt, der deinem Zielpunkt am nächsten liegt.

Ein weiterer denkbarer Ansatz wäre, daß der Mobot bei Punkt drei der Zielgeraden weiterhin folgt, sobald er diese schneidet. Die Effizienz diese Algorithmus wird meist gemessen durch den Quotient aus gefundener und kürzester Weglänge.
Dieser Ansatz kommt gänzlich ohne Wissen über seine Umgebung aus und erkundschaftet diese während sich der Roboter in ihr fortbewegt. Der gesamte Algorithmus arbeitet online, d.h. es wird kein Offline-Plan erstellt, dem der Mobot folgen kann.

7. Zusammenfassung

Schon lange spielen Roboter eine wichtige Rolle in der Industrie. Doch der immer noch stark anhaltende Trend zum tertiären Sektor, führte zu einer zunehmenden Entwicklung von mobilen Servicerobotern, die unbeaufsichtigt vielerlei Dienstleistungen erbringen sollen. So werden in amerikanischen Krankenhäusern bereits Roboter eingesetzt, die auf Anfragen selbständig Essen oder Medikamente holen und ausliefern.
Um einen solchen Mobot zu realisieren, benötigt man zuerst eine passende Hardware, welche allerdings ein Problem des Maschinenbaus und nicht der Künstlichen Intelligenz ist. Letztere ist bei der Navigations- und Bewegungsplanung wichtig, da die reale Welt viel komplexer und unberechenbare ist, als eine bloße Softwareumgebung. So müssen auch unvorhergesehene Probleme bewältigt werden.
Hierbei unterscheiden wir zwischen Navigation in bekannter und unbekannter Umgebung, wobei so-

wohl die Off-, als auch die Online-Kosten der Planung, berücksichtigt werden müssen. Ansätze wie die Zerlegung der Konfigurationsraums in Zellen und das Navigieren auf Skeletten vereinfachen die dreidimensionale Suche im kontinuierlichen Raum auf ein diskretes Graphensuchproblem. Problematisch ist dabei der Umgang mit fehlerhaften Sensordaten und Bewegungsfehlern, die das gelingen eines Plans verhindern können.

Als man feststellte, daß der klassische Ansatz aufgrund seiner Komplexität Echtzeitbedingungen nicht gewachsen war, wurde bis Mitte der 80er nur wenig Forschung auf dem Gebiet der Robotik und der künstlichen Intelligenz betrieben, welche jedoch trotzdem nie diesen klassischen Ansatz in Frage stellten. Die nächsten wichtigen Schritte und Weiterentwicklungen machten Stan Rosenschein und Rodney Brooks Mitte der 80er. Brooks verhaltensbasierte Robotik versuchte langes Planen und Erinnern an vorherige Aufräge durch bloßes Reagieren auf äußere Gegebenheiten, ähnlich dem menschlichen Reflex, zu ersetzen. Rosenscheins Situated Automata, der ein vollständiges Umweltmodell erstellt, liegt irgendwo zwischen klassischem und reflexgesteuerten Ansatz.

Doch obwohl sie später meist zu einem eigenständigen Forschungsbereich wurden, entstanden viele Gebiete der künstlichen Intelligenz durch spezielle Probleme der Robotik. Alles in allem kann man wohl sagen, daß auf diesem Gebiet noch viel Potential wartet. Angefangen bei experimentellen Antrieben bis hin zur perfekten Umwelterkennung und Wegplanungseffizienz, muß noch einiges an Forschung betrieben werden bis der perfekte Serviceroboter Wirklichkeit ist.

Quellen:

- *Artificial Intelligence- A Modern Approach* (Autoren, Jahr)
- *Vorlesungsskript Robotik*, Universtät Chemnitz (Dr. Johannes Steinmüller, WS 1998/99)
- *Vorlesungsskript Lernverfahren für mobile Roboter*, Universität Kaiserslautern (Dr.-Ing. Klaus-Werner Jörg, SS 1999)