Kollaboratives, verteiltes Problemlösen in unbekannten Umgebungen

- Masterarbeit -

im Studiengang Angewandte Informatik
der Fakultät Wirtschaftsinformatik und
Angewandte Informatik der
Otto-Friedrich-Universität Bamberg

Verfasser	:	**Andreas Hirschberger**
Themensteller	:	**Prof. Dr. Ute Schmid**
Abgabedatum:	:	**31. Juli 2007**

Inhaltsverzeichnis

Abbildungsverzeichnis

0.1 Einleitung

Immer mehr Aufgaben werden von automatisierten Systemen übrnommen. Die zunehmende Komplexität dieser Systeme erfordert neue Herangehensweisen. Monolithische Systeme sind zu statisch und oftmals zu komplex um umfassende Probleme effizient lösen zu können. Sie müssen nicht nur für spezielle Probleme entwickelt werden, sondern auch im aktiven Einsatz wartbar bleiben. Es muss möglich sein, das System weiter zu entwickeln und es sich ändernden Einsatzbedingungen anzupassen. Für den Benutzer muss es einfach zu bedienen und fehlertolerant sein. All diese Eigenschaften bringt die verteilte Agenten Architektur mit sich, da sich das komplexe System auf mehrere einfache, teilweise unabhängige Systeme aufteilt. Es ist jedoch nicht jedes Problem mit diesem Konzept effizient lösbar. Insbesondere die Frage nach der Qualität der erreichten Lösung ist von besonderem Interesse.

Auf dem Gebiet der Katastrophenforschung werden bereits Simulationen eingesetzt um mögliche Szenarien zu erzeugen und entsprechende Lösungs-Strategien zu entwickeln. Gerade in diesem Bereich ist der beste Weg zur erfolgreichen Bewältigung einer Katastrophe, aussreichend vorbereitet zu sein.

Die RobotRecueLeague, ein Teilbereich des Roboterwettkampfes RoboCup, beschäftigt sich mit diesen Simulationen. Darüber hinaus werden aber auch Systeme entwickelt, die mit den simulierten Szenarien umgehen und ohne Zutun des Menschen die Katastrophe bewältigen können. Ziel dieser Arbeiten ist es, automatisierte Systeme als Teil der Rettungsmassnahmen einzusetzen, um, beispielsweise in sehr gefährlichen Situationen, auch dann noch Mittel zur Verfügung zu haben, wenn keine Menschen mehr eingesetzt werden können. Auch die Organisation der Rettungskräfte, das Sammeln und Auswerten von Informationen sowie das Treffen von Vorhersagen über die weitere Entwicklung der Situation lassen sich durch den Einsatz von automatischen Assistenten vereinfachen.

Situationen wie brennende U-Bahnen sind extrem problematisch, da zum Einen durch Rauch und beengte Verhältnisse kaum Orientierungshilfen verfügbar sind und zum Anderen der Einsatz von menschlichen Rettern wegen akuter Lebensgefahr für die Retter selbst nicht möglich ist. In einem sol-

chen Fall könnten Roboter eingesetzt werden, um die Brände zu löschen und Eingeschlossenen und Verletzten Hilfe zu leisten. Diese Roboter sind möglicherweise von der Aussenwelt völlig abgeschnitten und nur in der Lage, untereinander Informationen auszutauschen.

Diese autonomen, möglicherweise spezialisierten Roboter sind ein ideales Beispiel für die Vorteile eines agentenbasierten, verteilten Systems, das durch Kooperation das gemeinsame Ziel - die Bewälltigung der Katastrophe - erfüllen kann.

Ein Agentensystem basiert auf Informationsgewinnung, -verarbeitung und -vermittlung. Entsprechend den gewonnenen Informationen entscheidet jeder Agent über die nächste Aktion, die er auszuführen beabsichtigt. Verschiedene Ansätze agentenbasierter Systeme treffen diese Entscheidungen anhand unterschiedlicher Grundsätze. In dieser Arbeit werden ausschliesslich zielgerichtete kognitive Agenten verwendet. Diese erzeugen aus dem erworbenen Wissen ein Modell ihrer Umgebung, anhand dessen sie die Konsequenzen ihres Handelns simulieren können. Dieses Modell ermöglicht eine umfassende Planung der zukünftigen Aktionen bis hin zur Lösung aller gestellten Probleme in der kürzest möglichen Zeit.

Bedingt durch die Unzugänglichkeit der Umgebung lassen sich nur eingeschränkt Modelle erzeugen und somit auch keine vollständige Planung durchführen. Die Agenten müssen entsprechend zunächst durch Erkundung Informationen sammeln, die zur Lösung zumindest einiger der gestellten Probleme ausreichen. Um die Erkundung zu erleichtern geben die Roboter ihre erworbenen Informationen an die anderen Agenten weiter.

Ist auf Grundlage der erworbenen Informationen Planung möglich, müssen die Agenten Wege finden, Konflikte in ihren Absichten zu erkennen und zu bewältigen und, soweit möglich, durch gezielte Verteilung der Teilziele und der Ressourcen, sowie Koordination der Handlungsabfolgen aller Agenten die Effizienz des Planes zu verbessern.

In dieser Arbeit wird ein Steuerprogramm entwickelt, das durch Kommunikation und Erkundung Informationen sammelt und, wann immer möglich, die Bearbeitung der Teilziele angeht. Die Bearbeitung aller Probleme wird durch Methoden der Koordination auf die Agenten aufgeteilt. Zum Vergleich

der verteilten mit der zentralen Planung wird zusätzlich eine zentrale Planung umgesetzt.

Kapitel 1

Einführung

In diesem Kapitel werde ich eine Einführung in die Grundlagen der Künstlichen Intelligenz geben. Zunächst werde ich das Prinzip und den Zweck von Agenten, sowohl Software-Agenten als auch embodied - verkörperlichte - Agenten erläutern, dann auf die notwendigen Prinzipien der zentralen und verteilten Planung eingehen und verschiedene Verfahren zur Erzeugung von Plänen vorstellen. Schliesslich werde ich die Grundlagen der Koordination vorstellen und den Umgang mit unbekannten Umgebungen erläutern.

1.1 Künstliche Intelligenz

1.1.1 Was ist Künstliche Intelligenz?

Die Fachdisziplin Künstliche Intelligenz hat sich in Forschung und Lehre etabliert und ist heute ein Bestandteil der Informatik mit interdisziplinärem Charakter. Die KI befasst sich sowohl mit der Konstruktion informationsverarbeitender Systeme, die „intelligente“ Leistungen erbringen, als auch mit der Modellierung menschlicher kognitiver Fähigkeiten mit Hilfe informationsverarbeitender Systeme. Forschung, Entwicklung und Anwendung der KI verfolgen das generelle Ziel, Vorgehensweisen der Informations-aufnahme und -verarbeitung zu entwerfen, die menschlichem Problemlösungsverhalten näher kommen, und daraus Methoden zur qualitativen Verbesserung und Ergänzung herkömmlicher Systeme der Informatik abzuleiten. Zu den bisher

erreichten Teilzielen gehören vor allem Programmsysteme auf höherem Abstraktionsniveau und, damit verbunden, neue Software-Techniken, die Erarbeitung formaler Grundlagen sowie verbesserte Formen der Mensch-Maschine-Kommunikation.

Im Verständnis des Begriffs „Künstliche Intelligenz“ spiegelt sich oft die aus dem Zeitalter der Aufklärung stammende Vorstellung vom „Menschen als Maschine“ wieder, dessen Nachahmung sich die so genannte starke KI zum Ziel setzt: eine Intelligenz zu erschaffen, die wie der Mensch kreativ nachdenken und Probleme lösen kann und die sich durch eine Form von Bewusstsein beziehungsweise Selbstbewusstsein sowie Emotionen auszeichnet. Die Ziele der starken KI sind auch nach Jahrzehnten der Forschung weiterhin visionär.

Anders als bei der starken KI geht es bei der schwachen KI darum, konkrete Anwendungsprobleme zu lösen. Insbesondere sind dabei solche Anwendungen von Interesse, zu deren Lösung nach allgemeinem Verständnis eine gewisse Form von „Intelligenz“ notwendig zu sein scheint. Letztlich geht es der schwachen KI somit um die Simulation intelligenten Verhaltens mit Mitteln der Mathematik und der Informatik; es geht ihr nicht um Schaffung von Bewusstsein oder um ein tieferes Verständnis von Intelligenz. Während die starke KI an ihrer philosophischen Fragestellung bis heute scheitert, sind auf der Seite der schwachen KI in den letzten Jahren bedeutende Fortschritte erzielt worden.

Neben den Forschungsergebnissen der Kerninformatik selbst sind in die KI Ergebnisse der Psychologie und Neurologie, Mathematik und Logik, Kommunikationswissenschaft, Philosophie und Linguistik eingeflossen. Die Beeinflussung der Neurologie hat sich in der Ausbildung des Bereichs Neuroinformatik gezeigt, der der biologieorientierten Informatik zugeordnet ist. Zusätzlich ist auch der ganze Zweig der Kognitionswissenschaft zu nennen, welcher sich wesentlich auf die Ergebnisse der künstlichen Intelligenz in Zusammenarbeit mit der kognitiven Psychologie stützt.

1.1.2 Wissensrepräsentation

Generell geht es der Künstlichen Intelligenz um die effiziente Sammlung, Speicherung und Nutzung von Informationen über die Umgebung in der sie arbeitet. Sie ist selbst Teil dieser Umgebung und muss daher auch Wissen über sich besitzen um bestimmte Aufgaben erfüllen zu können. Die Informationen, über die ein intelligentes System verfügt, müssen in einer dem Nutzungszweck des Systems angemessenen Form repräsentiert werden.

Praktisch kann man eine Wissensrepräsentation als die Abbildung eines Ausschnitts der Umgebung bezeichnen. Diese Abbildung, auch Abstraktion genannt, muss es dem intelligenten System ermöglichen, die darin enthaltenen Informationen automatisiert verarbeiten zu können. Das Wissen kann implizit im Programmcode gespeichert sein, oder explizit als Fakten.

Implizites Wissen

Implizites Wissen ganz Allgemein ist Wissen, das ein System besitzt, über das es jedoch nichts weiss. So sind z.B. Fähigkeiten wie Radfahren implizites Wissen, da es zwar verfügbar, nicht jedoch kommunizierbar ist. Im Falle intelligenter Softwaresysteme ist implizites Wissen im Programmcode explizit implementiert.

Explizites Wissen

Als explizites Wissen bezeichnet man jenes Wissen, das dem System selbst bewusst ist. Es wird auch als eindeutig kommunizierbares Wissen bezeichnet. Bei intelligenten Softwaresystemen sind beispielsweise Regeln zur Problemlösung, aber auch Faktenwissen in Form einer Datenbank explizites Wissen. Zusätzlich wird noch zwischen deklarativem und prozeduralem expliziten Wissen unterschieden:

Deklarativ: Es wird beschrieben, in welchem Verhältnis die betrachteten Einheiten in der realen Welt zueinander stehen. Eine getrennte, aktive Komponente kann dann dieses Wissen verwenden, um die Lösung zu berechnen. Der Vorteil liegt in der besseren Verständlichkeit und Lesbarkeit dieser

Methode. Der Aufwand, um eine konkrete Lösung zu berechnen, ist aber entschieden grösser.

Prozedural: Es wird Schritt für Schritt angegeben, wie die Aufgabe gelöst werden muss. Der Unterschied zu implizitem Wissen besteht darin, dass der Aufruf nicht explizit im Code vermerkt ist, sondern dass das System selbst erkennt, welches Wissen zur Lösung des Problems benötigt wird. Die explizite prozedurale Art, Wissen zu speichern, ist nicht leicht lesbar, aber effizienter in der Abarbeitung.

Anforderungen an die Wissensrepräsentationen

In der künstlichen Intelligenz muss die Wissensrepräsentation hohen Ansprüchen gerecht werden. Sie muss es ermöglichen, Wissen in einfacher, übersichtlicher Form schnell und effektiv zu verarbeiten.

Verarbeitbarkeit KI-Systeme müssen in der Lage sein, aus bestehendem Wissen durch logisches Verknüpfen auf neues Wissen zu schliessen. Die Wissensrepräsentation sollte das auf effiziente Weise unterstützen.

Expertensysteme versuchen z.B. aus bereits bekannten Fakten und aus Verknüpfungen von Fakten, den Regeln, auf neue Fakten zu schliessen, die dann wieder als bekanntes Wissen betrachtet werden.

Flexibilität Ein bestimmter Repräsentationsansatz soll einerseits geeignet sein, Wissen aus möglichst vielen Anwendungsbereichen (Domains) wie z.B. der Medizin oder der Geologie, darzustellen. Anderseits sollen unabhängig vom Anwendungsgebiet unterschiedliche Aufgabenstellungen, wie z.B. Diagnostik, Konstruktion und bzw. oder Simulation, effizient verwirklicht werden können.

So ist es das Ziel der Diagnose, ein bekanntes Muster, wie z.B. ein Krankheitsbild oder einen Fehler, wieder zuerkennen. Dabei wird die Lösung aus einer vorgegebenen Menge von Alternativen ausgewählt. Im Gegensatz dazu wird bei der Konstruktion die Lösung aus kleinen Bausteinen zusammengesetzt. Es gibt jedoch zu viele mögliche Kombinationen, um aus einer be-

grenzten Menge vorgegebener Alternativen auszuwählen. So kann z.B. bei der automatisierten Programmierung nicht aus der Menge aller möglichen Programme ausgewählt werden. Die Simulation unterscheidet sich von der Diagnose und der Konstruktion dadurch, dass kein vorgegebenes Ziel erreicht werden soll, sondern nur die Auswirkungen von Handlungen und Entscheidungen simuliert werden sollen.

Modularität Die Wissensbasis eines intelligenten Systems soll möglichst einfach veränderbar und erweiterbar sein. Am einfachsten ist das zu erreichen, wenn die Wissensbasis modular aufgebaut ist. Das bedeutet, spezielle Bereiche des Wissens, wie etwa Wissen über die Lösung eines bestimmten Teilproblems, sollen möglichst unabhängig vom Rest der Wissensbasis hinzugefügt oder verändert werden können.

Da bei deklarativem Wissen Informationen über die Zusammenhänge zwischen einzelnen Wissenselementen, z.B. wann und wo sie eingesetzt werden, nicht Bestandteil des Wissens selbst sind, kommt die deklarative Art, Wissen zu speichern, der Forderung nach Modularität entgegen.

Eine der Modularität verwandte Forderung ist die nach Trennung zwischen gespeichertem Wissen (der Wissensbasis) und dem für die Verarbeitung und Schlussfolgerung zuständigen Teil des Systems (der Inferenzmaschine).

Verständlichkeit Der Inhalt der Wissensbasis sollte gegenüber dem Benutzer leicht verständlich darstellbar sein. z.B. muss es für einen Benutzer nachvollziehbar sein, wie ein System zu einer bestimmten Entscheidung gekommen ist; er sollte aber auch einen Überblick über das gesamte gespeicherte Wissen erlangen können, um Lücken oder Widersprüche in der Wissensbasis des Systems aufdecken zu können. Viele Verfahren des automatisierten Wissenserwerbs erlauben es nur eingeschränkt, Zugriff auf die Daten zu erhalten. So sind insbesondere mathematische/statistische Lernverfahren auf Effizienz und nicht auf Verständlichkeit ausgelegt.

1.1.3 Verteilte künstliche Intelligenz

Verteilte künstliche Intelligenz bezeichnet Systeme, die die Eigenschaften der künstlichen Intelligenz besitzen und zusätzlich dezentral arbeiten. In traditioneller verteilter künstlicher Intelligenz geht es zunächst darum Verfahren der künstlichen Intelligenz durch ihre Verteilung auf mehrere Recheneinheiten schneller und somit nutzbarer zu machen. Im Grunde handelt es sich bei Systemen mit verteilter künstlicher Intelligenz also um mehrere Prozesse, die über gemeinsames Wissen verfügen und dieses für die Bearbeitung eines gemeinsamen Problems nutzen. In manchen Fällen ist eine Recheneinheit des Systems mit speziellen Schnittstellen ausgestattet, die dem Benutzer die Interaktion mit dem System ermöglichen.

Aus den Grundprinzipien verteilter künstlicher Intelligenz und verteilter Algorithmen hat sich letztlich die Agenten-basierte System-Architektur entwickelt.

1.2 Agenten

1.2.1 Was sind Agenten?

Bei Agenten handelt es sich um Bestandteile einer System-Architektur, in der die Bearbeitung eines Problems auf mehrere, mehr oder weniger spezialisierte, Agenten aufgeteilt ist. Dieses Konzept erleichtert die Entwicklung, Anpassung und Wartung eines komplexen Systems. Eine genaue Definition ist schwierig, da sich Agenten-basierte Systeme aus ihrer Anwendung entwickelt haben statt als zuerst entwickelter Entwurf. Oftmals wird ein Agent definiert als eine System, das die folgenden Eigenschaften hat:

- autonom - das Programm arbeitet weitgehend unabhängig von Benutzereingriffen
- proaktiv - das Programm löst Aktionen aufgrund eigener Initiative aus
- reaktiv - das Programm reagiert auf Veränderung seiner Umgebung
- sozial - das Programm kommuniziert mit anderen Agenten

- lernfähig/anpassungsfähig - das Programm lernt aufgrund zuvor getroffener Entscheidungen bzw. Beobachtungen.

Diese Eigenschaften treffen auch auf den Menschen zu. In einer Agenten-basierten Systemarchitektur kann auch der Mensch als „gleichwertiger“ Bestandteil einbezogen werden.

Das Merkmal „lernfähig“ ist jedoch keine notwendige Voraussetzungen um Bestandteil eines Agenten-basierten Softwaresystems zu sein. Wichtig für das Verständnis des Agentenprinzips ist vor allem der Begriff „autonom“. Jedoch lassen sich auch zu diesem bereits verschiedene Definitionen finden. Nach Wooldridge ist „Autonomie“ von Agenten:

> Agents are able to act without the intervention of humans or other systems: they have control both over their own internal state, and over their behaviour.

Der Begriff „sozial“ bezeichnet die Fähigkeit eines Agenten, mit anderen Agenten zu kommunizieren. Dies wird durch die Verwendung einer vereinheitlichten Sprache ermöglicht. Bedenkt man, dass gegebenenfalls auch menschliche Agenten Teil des Systems sein können, ist die Fähigkeit zur Kommunikation, besonders in einer verständlichen, definierten Form von grosser Bedeutung. Zur Lösung eines gemeinsamen Problems müssen Agenten in der Lage sein, zu kooperieren, um gemeinsame Ziele - unter Umständen auch auf Kosten eigener Interessen - zu erreichen. Diese Interessenabwägung wird oft in Zusammenhang mit der Fähigkeit, zu verhandeln, gebracht.

Bereits vor dem Auftauchen des „Agenten“-Begriffs erfüllten die meisten UNIX-Daemonen einige der oben genannten Anforderungen. Agenten-basierte Systementwicklung ist nach allgemeiner Auffassung eine logische Erweiterung des „*object-based concurrent programming paradigm*“.

Zusammengefasst ist ein Agent also eine Art Objekt, das die Kontrolle über seinen Zustand und sein Verhalten hat, mit anderen Agenten kommunizieren kann, auf Veränderungen der Umgebung reagiert und selbstständig neue Aktionen ausführt. Auf diese Weise organisiert sich ein Agenten basiertes System weitestgehend selbstständig. Das interessante an diesem Konzept

ist, dass Agenten eigenständig laufende Programme sind und das System somit einige nützliche Eigenschaften gewinnt:

- Das System ist unabhängig von der Implementierung, der Sprache und der Version der einzelnen Agenten.
- Jeder Agent ist nach belieben gegen einen semantisch gleichwertigen austauschbar.
- Das System lässt sich nach belieben auf verschiedene Rechner aufteilen; im Falle mobiler Agenten können die Agenten sogar zwischen mehreren Rechnern migrieren.

Wird der Begriff des Agenten im Zusammenhang mit künstlicher Intelligenz verwendet, dann meistens im Sinne von „intelligenten Agenten“. Intelligente Agenten zeichnen sich durch Wissen, Lernfähigkeit, die Fähigkeit Schlussfolgerungen zu ziehen sowie die Fähigkeit zu Verhaltensänderungen aus, also Fähigkeiten die dem menschlichen Verständnis nach Intelligenz voraussetzen oder sogar ausmachen.

Es gibt auch andere Begriffe, die oft in Zusammenhang mit Agenten auftauchen: Agenten können beispielsweise „mobil“ sein, d.h. den Ort ihrer Ausführung ändern. Beispielsweise migriert ein Agent auf den Rechner, auf dem benötigte Daten liegen, nutzt diese für seine Arbeit und kehrt dann auf den ursprünglichen Rechner zurück um das Ergebnis auszugeben. „Wohlwollen“ (*benevolence*) ist ein anderer den Agenten zugeschriebener Begriff, der besagt, dass die Ziele eines Agenten nicht widersprüchlich sein können. Der Agent wird daher immer die ihm gestellte Aufgabe bearbeiten. Im Hinblick auf die Wirtschaftlichkeit eines agentenbasierten Systems nimmt man des Weiteren an, dass jeder Agent rational ist, also der Erfüllung der Ziele nicht entgegen arbeitet.

1.2.2 Umgebung

Ein Schlüsselproblem der Agentenentwicklung ist, welche Aktion der Agent als nächstes ausführen soll, um sein Ziel erfüllen zu können. Die Methode

der Auswahl einer Aktion wird durch die Eigenschaften seiner Umgebung beeinflusst. Es lassen sich die folgenden Eigenschaften festlegen:

- zugänglich (*accessible*) / unzugänglich (*inaccessible*) - Zugänglich ist eine Umgebung, wenn der Agent alle relevanten Informationen über diese schnell und zuverlässig erhalten kann.

- deterministisch (*deterministic*) / nicht-deterministisch (*non-deterministic*) - In einer deterministischen Umgebung sind die Auswirkungen aller Aktionen eindeutig vorhersehbar.

- episodisch (*episodic*) / nicht-episodisch (*non-episodic*) - Eine Umgebung wird als „episodisch“ bezeichnet, wenn es keinen Zusammenhang zwischen dem Auftreten der einzelnen Ereignisse gibt. In solch einer Umgebung kann jedes Ereignis ohne Kenntnis früherer oder zukünftiger Ereignisse bearbeitet werden.

- statisch (*static*) / dynamisch (*dynamic*) - Eine statische Umgebung wird ausschliesslich durch Aktionen des Agenten verändert. In einer dynamischen Umgebung gibt es Veränderungen an der Umgebung, die sich der Kontrolle des Agenten entziehen.

- diskret (*discrete*) / kontinuierlich (*continuous*) - In diskreten Umgebungen gibt es nur eine feste, endlich Zahl von Zuständen und Aktionen. In kontinuierlichen Umgebungen kann es beliebig viele Zustände und somit auch Aktionen geben.

Die reale Welt stellt eine unzugängliche, nicht-episodische, nicht-deterministische, dynamische und kontinuierliche Umgebung dar und ist damit die denkbar anspruchsvollste Klasse von Umgebungen.

1.2.3 Agententypen

Agententypen werden entsprechend ihrer Architektur, also der Art und Weise, wie die Definition und Verwaltung des Agentenverhaltens erfolgt, in folgende Typen eingeteilt. Die Agentenarchitektur ist von den Eigenschaften

der Umgebung sowie den Anforderungen an den Agenten selbst abhängig. Zwar herrscht dabei leider eine grosse Begriffsvielfalt, aber die Einteilung in zwei weitgehend anerkannte Typ-Bereiche ist möglich:

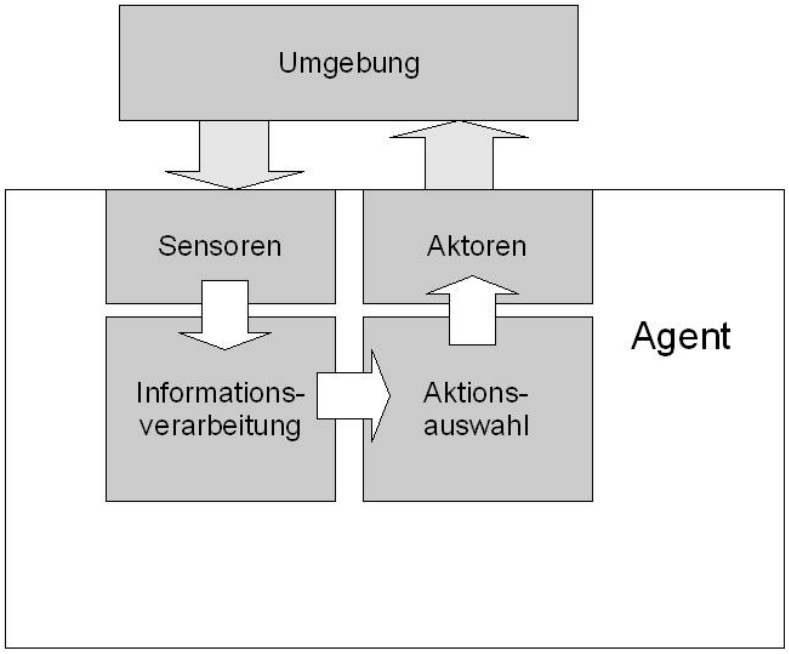

Abbildung 1.1: Struktur eines reaktiven Agenten

Reaktive Agenten Reaktive (bzw. subkognitive) Agenten verfügen prinzipiell nicht über eigenes Wissen, sondern agieren nur aufgrund ihrer Wahrnehmungen direkt und ohne Entscheidungsprozess.
Folgende Agententypen treten in diesem Zusammenhang öfter auf:

Einfach Reaktiver Agent ist der einfachste Typ. Der Agent erhält Sensorinformationen und wählt aufgrund von Bedingungs-Aktions-Regeln eine Aktion aus. Dies ist der Typ Agent, zu dem oftmals UNIX-Daemonen gerechnet werden.

Beobachtender Agent stellt eine Erweiterung des Einfachen Reaktiven

Agenten dar. Dieser Agententyp besitzt bereits ein Gedächtnis und sammelt Informationen über seine Umwelt sowie über die Auswirkungen seiner eigenen Aktionen auf sie. Die Bedingungs-Aktions-Regeln werden dann auf dieses Gesamtbild angewandt und nicht mehr nur auf die reinen Sensorinformationen.

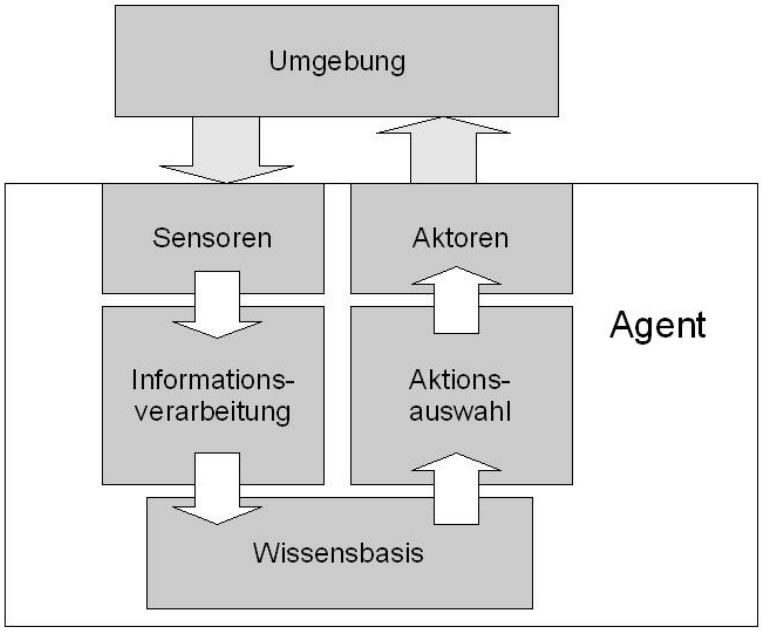

Abbildung 1.2: Struktur eines kognitiven Agenten

Kognitive Agenten Kognitive Agenten verwalten ein Modell ihrer Umwelt in einer eigenen Datenstruktur. Dadurch wird die Planung der Aktionen und schliesslich auch zielgerichtetes Handeln möglich. Eine bekannte Unterklasse ist die Agentendefinition in den BDI-Agenten-Architekturen durch Angabe der Annahmen(*Beliefs*), Bedürfnisse(*Desires*) und Absichten(*Intentions*).

Folgende Agententypen treten in diesem Zusammenhang öfter auf:

Zielbasierter Agent - Der Agent besitzt eine Zielvorgabe, die er zu erreichen versucht und entscheidet aufgrund seiner Sensorinformationen und

seines Wissens über die Folgen seiner Aktionen, welche Aktion ihn seinem Ziel am nächsten bringt. Da das Ziel nicht immer in einem Schritt erreicht werden kann, ist der Agent in der Lage, sein Vorgehen zu planen.

Nutzenbasierter Agent - Als Weiterentwicklung des zielbasierten Agenten besitzt der nutzenbasierte Agent ebenfalls eine Zielvorgabe. Es werden dabei alle möglichen und unmöglichen Zustände auf eine reelle Zahl abgebildet, welche den Nutzen für den Agenten repräsentiert. Dadurch ist er in der Lage, in Situationen, in denen mehrere Ziele erreichbar sind, zu entscheiden, welche Aktionen den grösseren Nutzen bringen, bzw. welche Ziele in der gegenwertigen Situation erstrebenswerter sind. Dies ist vor allem dann interessant, wenn nicht mit Sicherheit vorhergesagt werden kann, ob ein Ziel überhaupt erreicht werden kann. Der Agent kann damit eine Nutzen-einschätzung durchführen und gegebenenfalls zunächst ein anderes Ziel verfolgen.

1.2.4 Multi-Agenten-System

Bei einem Multi-Agenten-System oder MAS handelt es sich um ein System aus mehreren gleichartigen oder unterschiedlich spezialisierten, autark handelnden Einheiten, die gemeinsam ein Problem lösen. Im Allgemeinen sind damit Systeme aus „intelligenten Agenten“ gemeint. Man spricht dann auch von verteilter künstlicher Intelligenz oder DAI (distributed artificial intelligence).

Die Vorteile von Multi-Agenten-Systemen sind:

- Kosten/Nutzen - Um das selbe Problem zu lösen lassen sich Agenten mit höherer Spezialisierung einsetzen. Im Gegensatz zu monolithischen Systemen ist jedes spezialisierte Subsystem einfacher zu entwickeln, zu warten und zu verbessern.

- Flexibilität - Mit den selben, spezialisierten Agenten lässt sich durch Kombination eine Vielzahl von Problemen bearbeiten.

- Spezialisierung - Einzelne Agenten können hochgradig spezialisiert entwickelt werden. Die somit geringe Zahl der Handlungen (Aktionen) des

Agenten können leichter und gezielter optimiert werden.

- Unabhängigkeit - Die einzelnen Agenten sind nicht voneinander abhängig. Es müssen lediglich alle Fähigkeiten verfügbar sein, die für die Bearbeitung eines Problems erforderlich sind. Einzelne Agenten können leicht ausgetauscht und individuell verbessert werden.

Die genannten Vorteile sind sowohl für Forschung und Wissenschaft als auch für die kommerzielle Anwendung von besonderem Interesse, insbesondere da mit den selben Agenten Probleme bearbeitet werden können, ohne dass vorher eine genaue Beschreibung der Aufgabenstellung notwendig wäre.

Multi-Agenten-Systeme haben viele der oben aufgeführten Eigenschaften mit Objekt-Orientierten-Systemen gemein, von denen sie sich aber in einigen Punkten unterscheiden. Objekt-Orientierte-Softwaresysteme sind geschaffen, um ein bestimmtes Problem zu bearbeiten. Zu diesem Zweck nutzen Objekte andere Objekte, sie geben Aufträge oder werden von anderen Objekten genutzt. Agenten hingegen haben eigene Interessen bei der Bearbeitung gemeinsamer Ziele. So kann ein Agent durchaus auch die Zusammenarbeit verweigern, wenn er davon keinen Nutzen hat. Objekte sind also eher Teil einer - möglicherweise verteilten - Software, während Agenten Teil eines kooperativen Systems zur Lösung eines Problems sind.

1.2.5 Roboter Agenten

Roboter (auch: *embodied*) Agenten sind Agenten, deren Aktionen durch entsprechende mechanische Hilfsmittel Einfluss auf die physische, reale Welt ausüben. Der Agent kann beispielsweise seine Position im Raum verändern oder mittels eines Greifarms Gegenstände manipulieren. Da das grundlegende Konzept des Agenten, der seine Umgebung wahrnimmt und zielgerichtet entscheidet, welche Aktion als nächste auszuführen ist, ideal auf Roboter anwendbar ist, gewinnt diese Form der Steuerung immer mehr Bedeutung. Auch die Darstellung des Agenten als Teil einer Umgebung, die er mittels Aktionen verändert, ist wie für Roboter geschaffen. Tatsächlich verdankt die Agentenbasierte Systemarchitektur der Idee vom Roboter als einem Agenten

in einem physischen System einige bedeutende Beiträge. Gerade in dem Bereich der intelligenten Agenten, der planenden Agenten und der dynamischen Planung sind praktische Anwendungen im Bereich der Robotik vorhanden.

Da Roboter-Agenten in der realen Welt agieren, ist ihre Umgebung grundsätzlich als unzugänglich, nicht-episodisch, nicht-deterministisch, dynamisch und kontinuierlich anzunehmen. Dies erschwert die Wahrnehmung, Datenhaltung und Planung. Daher werden heute Roboter-Agenten in erster Linie in kontrollierter, leicht abstrahierbarer Umgebung eingesetzt. Will man den Agenten in realistischen Situationen einsetzen, so muss er in der Lage sein, Veränderungen seiner Umgebung in sein Modell der Welt aufzunehmen, und zwar unabhängig davon ob er diese Veränderungen selbst herbeiführt oder ob sie nicht durch ihn beeinflusst eintreten.

1.3 Planung

1.3.1 Grundlagen

Planung ist laut allgemeiner Definition der systematische Prozess, mit dem ein Verfahren oder eine Handlungsabfolge zum Erreichen eines Zieles erzeugt wird. In der künstlichen Intelligenz wird Planung eingesetzt um einem Akteur die gezielte und möglichst effiziente Lösung eines Problems zu ermöglichen.

Im Falle der Agentenarchitektur, in der jeder Agent Akteur ist, wird häufig zwischen reaktiven und planenden Agenten unterschieden. Der reaktive Agent stützt seine Entscheidungen dabei nur auf seine derzeitigen Sensorwerte. Zum Lösen vieler Probleme ist dies ausreichend. Beispielsweise lässt sich Navigation sowohl reaktiv als auch geplant durchführen. Im reaktiven Fall wäre der navigierende Agent mit der Information ausgestattet, in welcher Richtung das Ziel liegt. Zusätzlich stehen ihm aktuelle Abstandssensordaten zur Verfügung, um Hindernisse erkennen und Kollisionen vermeiden zu können. Es lassen sich jedoch leicht Szenarien finden, in denen der Agent sein Ziel auf diese Weise nicht erreichen kann.

Planende Agenten hingegen bestimmen vor Beginn der Aktivitäten zur

Problemlösung eine Abfolge von Handlungen, welche das gestellte Ziel garantiert erfüllen, sofern es erfüllbar ist. Die Eigenschaft eines Plans, durch Ausführen der geplanten Handlungsschritte in der geplanten Reihenfolge das gesteckte Ziel immer zu erfüllen, nennt man Korrektheit oder Validität eines Planes. Natürlich ist nicht jedes Problem mit den verfügbaren Mitteln lösbar und somit muss es auch nicht immer einen validen Plan geben. Das zur Planung eingesetzte Verfahren muss daher in der Lage sein, feststellen zu können, ob das Problem überhaupt lösbar ist. Diese Eigenschaft des planenden Systems nennt man Vollständigkeit. Jeder tatsächlich nutzbare Planungsalgorithmus muss vollständig sein und einen validen Plan erzeugen.

Nicht für jedes Planungsproblem ist also tatsächlich ein valider Plan erzeugbar. Es lässt sich zeigen, dass es keinen „*general planning problem solver*“ geben kann. Es gibt jedoch Verfahren für die sogenannten klassischen Planungsprobleme. Da im Rahmen dieser Arbeit unlösbare Probleme nicht interessieren, betrachten wir im Folgenden nur die klassischen Planungsprobleme und Ansätze zu ihrer Lösung. Mathematisch betrachtet handelt es sich bei einer Handlung (Aktion) um die Anwendung einer Übergangsfunktion, die eine Menge von Eigenschaften (Zustand) auf eine andere abbildet. Der gesuchte Plan ist daher der Satz von Aktionen, der von einem festgelegten Ausgangszustand zu einem gewünschten Zielzustand führt. Jedes klassische Planungsproblem lässt sich also als Zustandsübergangssystem betrachten.

Ein Zustandsübergangssystem ist ein Quadrupel $\Sigma = (Z, A, E, \gamma)$ wobei

$Z = \{z_1, z_2, ...\}$ eine endliche Menge von Zuständen
$A = \{a_1, a_2, ...\}$ eine endliche Menge von Aktionen
$E = \{e_1, e_2, ...\}$ eine endliche Menge von Ereignissen
$\gamma = Z \times A \times E \rightarrow 2^Z$ eine Zustandsübergangsfunktion

Das Zustandsübergangssystem lässt sich als gerichteter Graph darstellen, in dem die Zustände Knoten und die Zustandsübergänge Kanten sind. In diesem - Zustandsgraph oder Zustandsraum genannten - Graphen lassen sich mittels Verfahren zur Pfadsuche in Graphen Aktionsfolgen, also Pläne, finden, die von einem definierten Ausgangszustand zum Ziel führen.

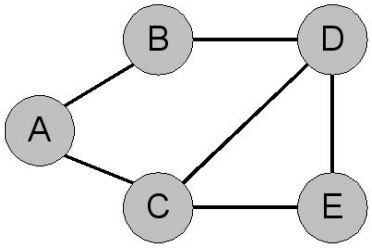

Abbildung 1.3: Zustandsraum als Graph

Klassische Planungsprobleme lassen sich also wie folgt definieren:

Gegeben:

- eine Beschreibung der bekannten Bestandteile des Ausgangszustands der Welt (in einer formalen Sprache, üblicherweise propositional Logic), bezeichnet als I,
- eine Beschreibung des Ziels (bzw. mehrere Zielzustände), bezeichnet als G und
- eine Beschreibung der verfügbaren (atomaren) Aktionen in Form von Zustandsübergängen,

legen einen Plan fest, der jeden dem Ausgangszustand entsprechenden Zustand auf einen möglichen Zielzustand überführt.

Beispiel

Stellt man sich einen Roboter vor, der sich mittels Räder bewegen und mit einem Greifarm Gegenstände aufheben und auf eine Transportfläche laden kann, so erhällt man die Aktionen `move`, `pick_up`, `put_down`. Wobei

- `move(x, y)` mit $x, y \in \{A, B, C\}$ - bewegt den Roboter von einer Position `x` zu einer Position `y`. Um diese Aktion anwenden zu können muss der Roboter vorher an der Position `x` sein. Nach Anwendung dieser Aktion ist sichergestellt, dass `roboter(x)` nicht mehr, `roboter(y)` hingegen ab jetzt gilt.
- `pick_up(g)` - hebt einen Gegenstand auf und lädt ihn auf die Ladefläche. Damit diese Anweisung ausgeführt werden kann muss vorher

`roboter(x)` und `gegenstand(y)` mit `x == y` gegolten haben. Nach Anwendung der Aktion gilt `gegenstand(y)` nicht mehr, dafür gilt `gegenstand(ladeflaeche)`.

- `put_down(g)` - legt einen auf der Ladefläche befindlichen Gegenstand auf den Boden. Hierzu muss vor der Anwendung der Aktion `gegenstand(ladeflaeche)` und `roboter(x)` mit $x \in \{A, B, C\}$ sein. Nach der Anwendung gilt nicht mehr `gegenstand(ladeflaeche)` sondern `gegenstand(x)`.

Ein möglicher initialer Zustand könnte dann dieser Roboter an einem Ort A sein und ein Gegenstand O an einem anderen Ort B. Das Ziel ist hier, dass sich am Ende der Gegenstand am Ort C befindet.

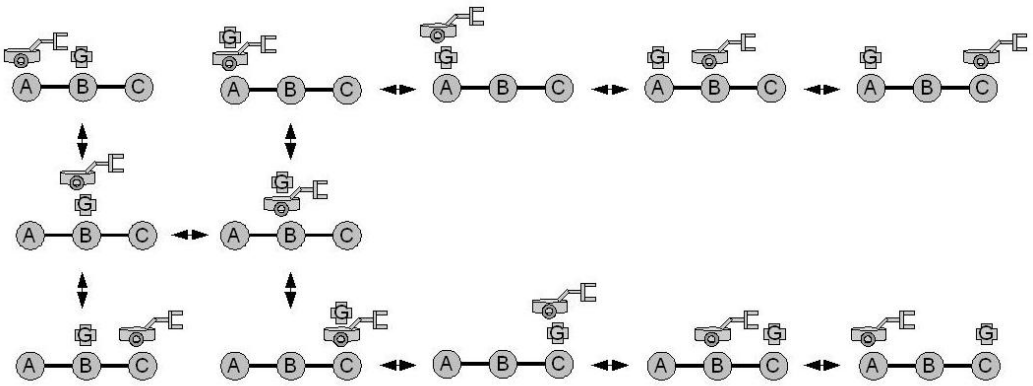

Abbildung 1.4: Zustandsraum eines Transportroboters

Im Allgemeinen lässt sich der Zustandsraum eines Planungsproblems jedoch nicht vollständig konstruieren, da seine Grösse schnell die verfügbaren Kapazitäten bezüglich Speicher und Rechenleistung überschreitet. Dies ist für die gängigen Verfahren aber auch nicht erforderlich. Normalerweise lassen sich stets die von einem gegebenen Zustand aus erreichbaren Nachfolge-Zustände bei Bedarf berechnen. Das Berechnen aller auf einen Zustand folgender Zustände nennt man Expansion eines Zustands. Da die Suche stets von einem Zustand aus beginnt und Nachfolger erzeugt, welche dann ihrerseits expandiert werden bildet jede Suche einen Baum, den sogenannten

Suchbaum. Dieser stellt nur einen Ausschnitt des gesamten Zustandsraumes dar. Ziel vieler spezialisierter Suchverfahren ist es, diesen Suchbaum soweit wie möglich einzuschränken und die Suche so zu beschleunigen.

Zusätzlich wird bei der Suche zwischen informierter und uninformierter Suche unterschieden. Informierte Suche bezeichnet dabei einen Algorithmus, der den Suchraum unter Verwendung von verfügbarem Wissen zielgerichtet durchsucht, während uninformierte Suchverfahren den Baum blind durchsuchen.

1.3.2 STRIPS

STRIPS (Stanford Research Institute Problem Solver) ist ein Automatisches Planungssystem, welches von Richard Fikes und Nils Nilsson 1971 vorgestellt wurde. Den selben Namen trägt auch die Formale Sprache, in der Probleme beschrieben und dem Algorithmus zugeführt werden. Diese Sprache ist die Grundlage der meisten heute genutzten Beschreibungssprachen für Planungssysteme.

Zustände STRIPS verwendet zur Beschreibung eines Planungsproblems eine Teilmenge der Prädikatenlogik. In dieser werden Fakten, also atomares Wissen über einen Zustand der Welt, als Prädikat einer festen Stelligkeit und Konstanten beschrieben. $at(Roboter, A)$ wäre in dem oben beschriebenen Beispiel etwa die Information, dass sich der Roboter an der Position A befindet. Ein vollständiger Zustand ist eine Menge solcher Fakten. Der oben beschriebene Initial-Zustand liesse sich darstellen als $S = at(Roboter, A), at(Gegenstand, B)$. Die Beschreibung des Zielzustands enthält nur die relevanten Fakten. Ist ein Teilzustand nicht explizit angegeben, ist sein Zustand nicht von Bedeutung. Der Zielzustand wäre dargestellt als $Z = at(Gegenstand, C)$

Zustandsübergänge In STRIPS werden aus Aktionen, die von einem Zustand zu einem anderen überführen, zunächst Operatoren, die Variablen enthalten können und somit mehrere Aktionen repräsentieren. Ein solcher Ope-

rator ist durch drei Komponenten charakterisiert.

- Operatorsymbol o (Name des Operators) gefolgt von einer Liste von n Termen, die in Vorbedingungen und Effekten Verwendung finden. n ist die feste Stelligkeit des Operators.
- eine Menge von Literalen $prec(o)$, den Vorbedingungen (preconditions)
- eine Menge von Literalen $eff(o)$, den Effekten

Enthält ein Operator zusätzlich Variablen, so ist er partiell instantiiert. In der Regel wird die Menge der Effekte $eff(o)$ in zwei disjunkte Mengen aufgeteilt:

- $add(o)$ - Die Liste der Fakten, die nach Anwendung des Operators „Wahr“ sind
- $del(o)$ - Die Liste der Fakten, die nach Anwendung des Operators nicht mehr gelten

Die Aktionen *bewegen*, *aufheben* und *abladen* werden in STRIPS zu den Operatoren:

```
move ?Akteur ?Von ?Nach
prec: at(?Akteur, ?Von)
add:  at(?Akteur, ?Nach)
del:  at(?Akteur, ?Von)

pick_up ?Akteur ?Gegenstand ?Wo
prec: at(?Akteur, ?Wo), at(?Gegenstand, ?Wo)
add: at(?Gegenstand, ?Akteur)
del: at(?Gegenstand, ?Wo)

put_down ?Akteur ?Gegenstand ?Wo
prec: at(?Akteur, ?Wo), at(?Gegenstand, ?Akteur)
add: at(?Gegenstand, ?Wo)
del: at(?Gegenstand, ?Akteur)
```

Diese teilinstantiierten Operatoren lassen sich im Zusammenhang mit einem konkreten Zustand vollständig instantiieren. Im vorliegenden Beispiel könnte durch „matchen“ der Fakten des Initial-Zustands auf Variablen der Vorbedingungen beispielsweise der Operator `move ?Akteur ?Von ?Nach` zur Aktion `move Roboter A B` werden. Diese Aktion ist auf den betrachteten Zustand anwendbar, das heisst, dass die in den Vorbedingungen geforderten Fakten in diesem Zustand gelten.

Ein möglicher Plan für dieses Beispiel ist in der Abbildung 1.5 dargestellt.

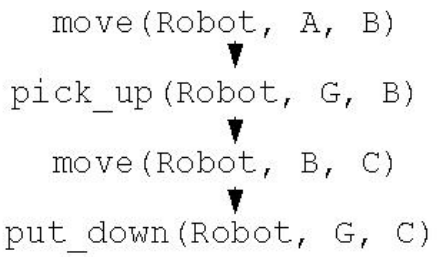

Abbildung 1.5: Möglicher Lösungsplan

1.3.3 Total order plan

Die gängigsten Planungsverfahren erzeugen einen „*total order plan*“ also einen vollständig geordneten Plan, der eine exakte, feste Abfolge von Aktionen definiert. Der Vorteil eines solchen Plans ist, dass man ihn dem Agenten zur Verfügung stellen und sich auf seine dem Plan exakt folgende Bearbeitung verlassen kann. Es ist also nicht nur die Erfüllung der gestellten Ziele, sondern auch der Weg ihrer Bearbeitung vorhersagbar.

Beispiel: Erweitert man das oben beschriebene Beispiel um einen zweiten Gegenstand $G2$, der sich ebenfalls an der Position B befindet, fügt also $at(G2, B)$ hinzu, erhält man den in Abbildung 1.6 dargestellten Plan vollständiger Ordnung.

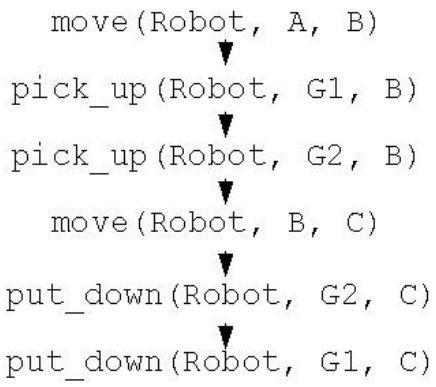

Abbildung 1.6: Vollständig geordneter Plan

1.3.4 Partial order plan

Von grossem Interesse für viele weiterführende Arbeiten mit Plänen, insbesondere für die Verteilung von zentral geplanten Aufgaben auf verschiedene Akteure oder für die Verfeinerung von Metaplänen, also Plänen, die nicht exakte Aktionen, sondern vielmehr Aufgaben, die zur Lösung des Problems führen enthalten, sind „teilweise geordnete Pläne“. Bei diesen werden Aktionen, die in beliebiger Reihenfolge ausgeführt werden können, zusammengefasst. Zur Ausführung müssen diese Aktionen dann wieder in eine bestimmte Reihenfolge gebracht werden. Diese Sequentialisierung kann randomisiert oder reaktiv durchgeführt werden. Die Aktionen können also in beliebiger Reihenfolge durchgeführt werden oder ihre Reihenfolge richtet sich nach Informationen, die erst kurzfristig feststehen und die Ausführbarkeit der anderen Aktionen nicht beeinflussen. Betrachtet man eine Einkaufsliste als Teil eines „teilweise geordneten Plans“, ist es einleuchtend, dass sich die tatsächliche Reihenfolge der Warenentnahme nach der Position der Dinge im Laden richtet, was für die Aufgabenstellung selbst aber völlig irrelevant ist.

Beispiel: Ein „teilweise geordneter Plan“ für das obige Szenario ist in der Abbildung 1.7 dargestellt.

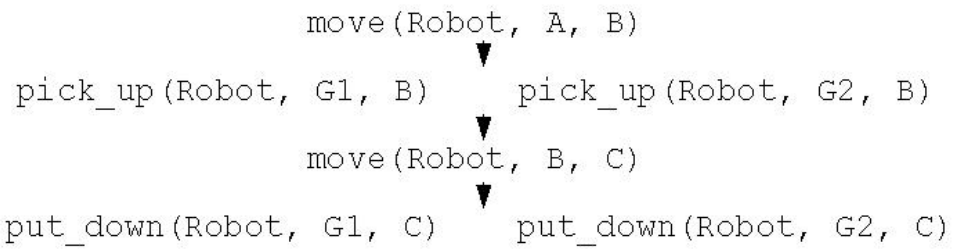

Abbildung 1.7: Teilweise geordneter Plan

1.3.5 Suchverfahren

Im Folgenden werde ich die gängigsten Suchverfahren vorstellen. Sie basieren auf einem systematischen Durchwandern des durch das Problem aufgebauten Zustandsraumes. In diesem ist jeder Zustand ein Knoten. Die einzelnen Zustände werden durch Zustandsübergangsfunktionen als Kanten verbunden. Dies bildet einen Graphen, der mit Mitteln der Graphentheorie analysiert werden kann. Von besonderem Interesse ist dabei die *shortest path* Suche. Aus Gründen der Problemkomplexität können die Verfahren nicht auf den gesamten Suchraum angewandt werden. Die folgenden Verfahren nutzen einen erst während der Anwendung des Verfahrens aufgebauten Suchraum. Dabei bestimmt die Art und Weise wie dieser SSuchbaumërzeugt wird über die *performance* des Verfahrens. Algorithmisch arbeiten alle Verfahren nach dem selben Grundkonzept:

1. Startknoten expandieren - Nachfolger speichern

2. Nachfolgeknoten zur Expansion auswählen

3. Knoten expandieren - Nachfolger speichern

4. Wenn der Zielknoten gefunden oder kein Knoten mehr zur Expansion verfügbar ist endet der Algorithmus sonst weiter bei 2.

Breitensuche

Die Breitensuche expandiert den ausgewählten Knoten und speichert alle konstruierten Nachfolger in einer FIFO-Queue. Der nächste zu expandierende Knoten wird dann vorne aus der Warteschlange herausgenommen. Auf diese Weise werden nach dem Startknoten zunächst alle Knoten der ersten Ebene des Suchbaums expandiert, dann alle der zweiten Ebene usw. Da dieses Verfahren alle möglichen Pfade einer bestimmten Länge untersucht, bevor es die nächste Ebene angeht, findet es garantiert den kürzesten Pfad, ist also optimal. Da es den Suchraum systematisch durchsucht findet es auch garantiert einen Lösungspfad, wenn einer existiert, ist somit vollständig. Der grösste Nachteil dieses Verfahrens ist der grosse Speicherbedarf, da jeder gefundene Knoten gespeichert werden muss.

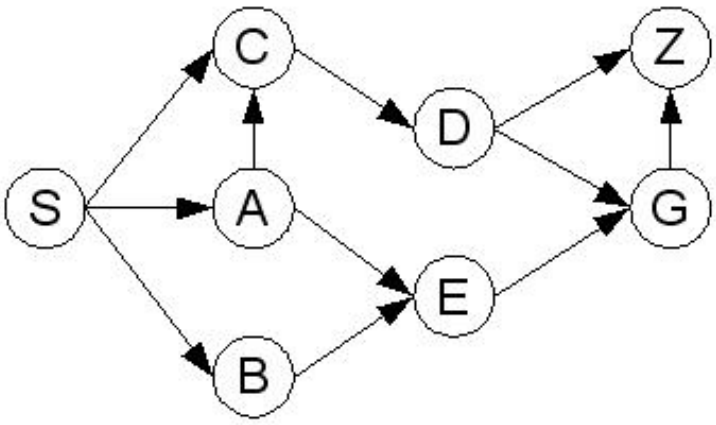

Abbildung 1.8: Problemgraph

Beispiel: Betrachtet man den in Abbildung 1.8 dargestellten Graphen, wobei S der Startknoten der Suche und Z das Ziel ist, ergibt die abgschlossene Suche den in Abbildung 1.9 dargestellten Suchbaum. Am Ende wurde der Pfad S $\rightarrow$ C $\rightarrow$ D $\rightarrow$ Z gefunden. Dieser Pfad ist optimal, es gibt also keinen kürzeren.

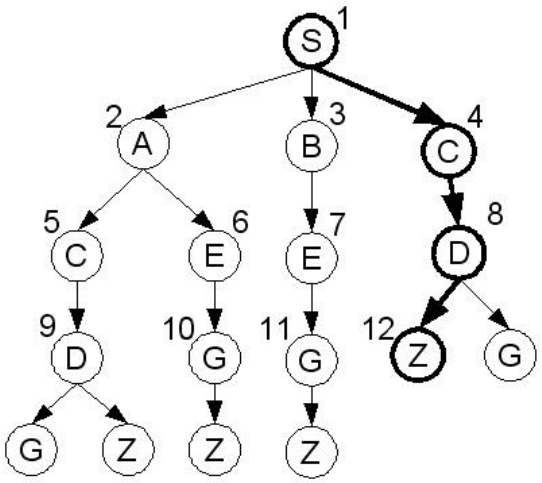

Abbildung 1.9: Suchbaum der Breitensuche

Tiefensuche

Die Tiefensuche expandiert den ausgewählten Knoten und speichert alle konstruierten Nachfolger in einem LIFO-Stack. Der nächste zu expandierende Knoten wird dann oben vom Stapel genommen. Auf diese Weise wird der Suchbaum zunächst in die Tiefe durchsucht, bis ein Blatt erreicht wird. Dieses Verfahren ist nicht vollständig, da der Suchbaum nicht endlich sein muss, sondern beispielsweise Zyklen enthalten kann. Die Suche ist auch nicht optimal, da stets die Möglichkeit besteht, dass ein noch kürzerer Pfad existiert. Führt man einen Zyklentest bei der Expansion von Knoten ein, lässt sich zumindest die Vollständigkeit wieder herstellen. Mit diesem ist die Tiefensuche im aufwändigsten Fall vom Laufzeitaufwand $O(|V|+|E|)$ mit $|V|$ für die Anzahl der Knoten und $|E|$ für die Anzahl der Kanten. In manchen Fällen bietet die Tiefensuche eine gute Chance, eine Lösung wesentlich schneller zu finden als die Breitensuche. Der Vorteil dieses Verfahrens ist der vergleichsweise geringe Speicherbedarf, da nur die Knoten eines Pfades gespeichert werden.

Beispiel: Wendet man auf das obige Beispiel die Tiefensuche an, so erhält man möglicherweise den in Abbildung 1.10 dargestellten Suchbaum. Der so

gefundene Pfad S → A → C → D → G → Z ist länger als der mittels Breitensuche gefundene Pfad. Dafür waren allerdings auch weniger Expansions-Schritte notwendig, um diesen zu finden.

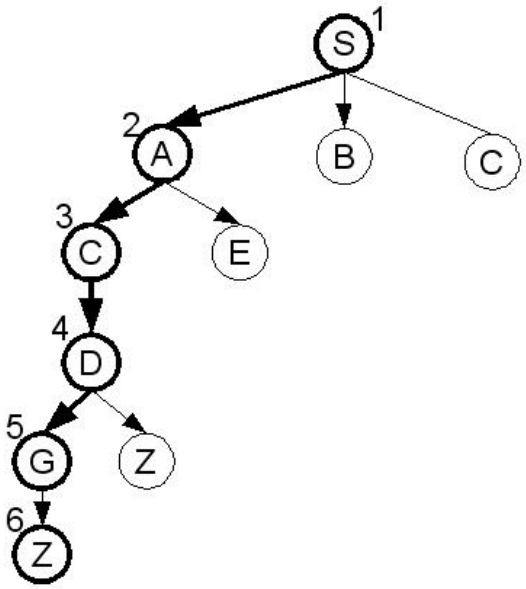

Abbildung 1.10: Suchbaum bei Tiefensuche

Iterative Tiefensuche

Die Iterative Tiefensuche basiert auf der Tiefensuche. Um die Optimalität zu gewährleisten wird die Suchtiefe begrenzt. Auf diese Weise wird, falls es keine Lösung innerhalb dieser Tiefe gibt, der gesamte, eingeschränkte Suchbaum mittels Tiefensuche erfolglos durchsucht. Dann wird die Suchtiefe erhöht und die Suche erneut durchgeführt. So werden Pfade ihrer Länge nach geordnet durchsucht, was letztlich zu einem optimalen Pfad führt. Der Aufwand ist dabei nach wie vor $O(|V|+|E|)$, die Vorteile der Tiefensuche bezüglich Speicherplatz gelten ebenfalls. Somit ist dieses Verfahren für das Durchsuchen grosser Zustandsräume besser geeignet als Breiten- oder Tiefensuche.

Beispiel: Auf den in Abbildung 1.8 definierten Graphen soll eine iterative Tiefensuche angewendet werden. Der sich für jeden Iterationsschritt daraus

ergebende Suchbaum ist in Abbildung 1.11 dargestellt. Das Ende der Iterativen Tiefensuche ist erreicht, wenn der optimale Pfad S → C → D → Z gefunden ist. Für dieses Beispiel ist dieses Verfahren deutlich aufwendiger als die beiden anderen.

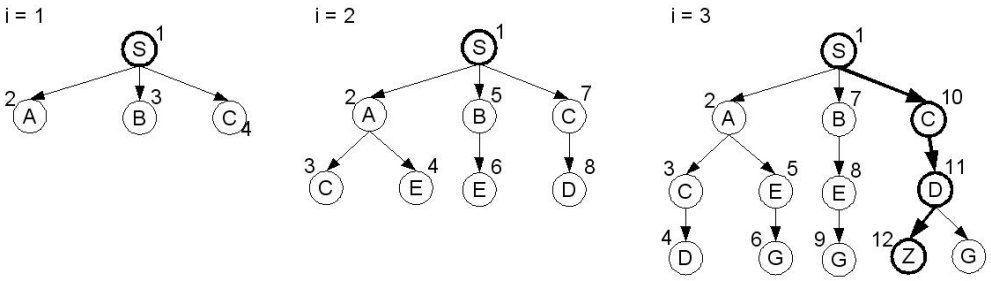

Abbildung 1.11: Suchbäume bei iterativer Tiefensuche

Branch-and-Bound Suche

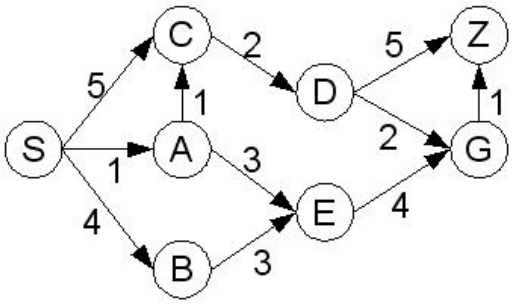

Abbildung 1.12: Problemgraph mit Kosten

Werden den Kanten eines Graphen Kosten zugewiesen, die den Aufwand des Übergangs von einem Zustand zu einem anderen widerspiegeln, lassen sich andere Verfahren einsetzen, die auf den Prinzipien der Breiten- bzw. Tiefensuche basieren. Die Verwendung solcher Aufwandskosten ist nicht in jedem Szenario möglich; in der allgemeinen Aktionsplanung sind keine In-

formationen über Kosten vorgesehen, sie stellen eine Erweiterung der klassischen Planung dar. Die ideale Anwendung von kostenbasierter Suche ist die Wegplanung, da die Kosten direkt aus der Distanz zweier Orte - und gegebenenfalls weiteren Einflüssen wie z.B. der Streckenbeschaffenheit - bestimmt werden können.

Die Idee, die hinter der Branch-and-Bound Suche steht, ist folgende: Angenommen, man weiss von einem Pfad, dass er vom Start zum Ziel führt und möchte überprüfen, ob es sich um den optimalen Lösungspfad handelt, so genügt es, alle anderen Pfade nur soweit zu betrachten, bis deren Kosten grösser werden, als die Kosten der Benutzung des optimalen Pfades.

Dieser Suchalgorithmus basiert auf der Nutzung einer sortierten Liste als Ablage der zu erweiternden Pfade. Jedem dieser Pfade ist dabei der Wert zugeordnet, der den Kosten seiner Nutzung entspricht. Die Pfade sind nach diesem Wert aufsteigend sortiert, der bislang günstigste Pfad ist daher vorne in der Liste. Dieser wird aus der Liste entfernt, um die Nachfolger seines letzten Knotens erweitert - dabei entstehen neue Pfade mit einem gemeinsamen Pfadkopf - und wieder in die Liste einsortiert. Endet der erste Pfad in der Liste auf dem Zielzustand, ist der kürzeste Pfad gefunden. Dass es der kürzeste optimale Pfad ist, liegt daran, dass alle anderen Pfade, die auf dem Zielzustand enden, entweder wegen ihrer höheren Kosten tiefer in der Liste stehen oder durch Expansion bereits teurerer Pfade erzeugt werden. Die Tatsache, dass ein zur Expansion ausgewählter Pfad stets der kürzestmögliche zu seinem letzten Zustand ist, lässt sich zur Beschleunigung des Verfahrens nutzen, indem für jeden Knoten eine ëxpandiertİnformation gespeichert wird, sobald er einmal Endknoten eines zur Expansion ausgewählten Pfades war. Dieser Knoten wird dann nicht mehr bei der Expansion beachtet. Diese Eigenschaft nutzt beispielsweise der Dijkstra-Algorithmus zur Erzeugung aufspannender Bäume von Graphen.

Beispiel: In Abbildung 1.12 wird hierzu der Graph aus Abbildung 1.8 um die Kosten erweitert, die der Übergang von einem Knoten zu einem anderen verursacht. Wendet man nun die Branch-and-Bound Suche an, erhält man den in Abbildung 1.13 dargestellten Suchbaum. Der unter Berücksichtigung der Kosten günstigste Pfad ist nun S $\rightarrow$ A $\rightarrow$ C $\rightarrow$ D $\rightarrow$ G $\rightarrow$ Z.

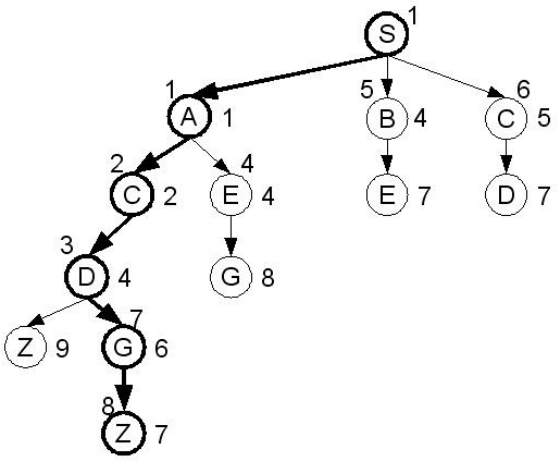

Abbildung 1.13: Suchbaum bei Branch-and-Bound Suche

A* Algorithmus

Heuristik
nach Z von

S:	6
A:	3
B:	8
C:	4
D:	2
E:	5
G:	1

Abbildung 1.14: Heuristik für das Suchproblem

Erweitert man den Branch-and-Bound-Algorithmus um geschätzte - heuristische - Informationen über die Restkosten erhält man den effizientesten Suchalgorithmus A*. Dabei wird derjenige Knoten als nächstes expandiert, bei dem die erwarteten Gesamtkosten am geringsten sind. Die Gesamtkosten setzen sich aus den bisherigen tatsächlichen Kosten eines Pfades und den erwarteten Restkosten zusammen.

Ein Problem bei diesem Suchverfahren ist das Finden und die Auswahl geeigneter Heuristiken. Um funktionieren zu können muss die Heuristik die tatsächlichen Restkosten immer unterschätzen. Für Pfadplanung im karte-

sischen Koordinatenraum eignet sich beispielsweise die Luftlinie zwischen zwei Knoten als Heuristik, da ein tatsächlicher Pfad immer wenigstens so lang sein muss wie diese Strecke. Eine Heuristik ist immer eine relaxierte Variante des eigentlichen Problems. Es werden also Einschränkungen und Bedingungen des Problems ausser acht gelassen. Für allgemeine Planung bei klassischen Planungsproblemen eignet sich eine Modifikation der Operatoren, bei der die del-Effekte weggelassen werden. Somit lässt sich ein Plan finden, der, unter Ausserachtlassung von gegenseitigen Störungen der Aktionen, auf schnellstmögliche Weise zur Lösung des Problems führt. Die Anzahl der benötigten Aktionen dient als Heuristik zur Bewertung möglicher Nachfolger.

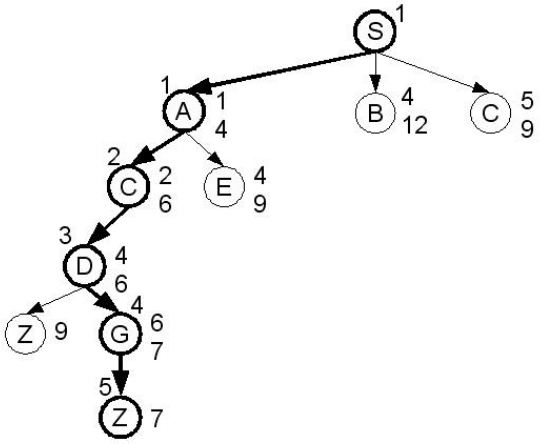

Abbildung 1.15: Suchbaum bei A*

Beispiel: Wendet man A* auf den obigen Graphen an indem man die in Abbildung 1.14 definierten Heuristiken einsetzt erhält man den in Abbildung 1.15 dargestellten Suchbaum. Der ermittelte Pfad S → A → C → D → G → Z ist optimal und es lässt sich zeigen, dass auch der durchsuchte Suchbaum minimal ist. Es kann also kein Verfahren existieren, das durch Suche schneller den optimalen Pfad findet.

1.3.6 Planungsverfahren

Grundlagen

Grundsätzlich lassen sich, je nach Problemstellung, Verfahren einsetzen, die auf den oben beschriebenen Suchverfahren basieren. Man unterscheidet dabei zwischen vorwärts- und rückwärtsgerichteter Planung. Eine weitere grundsätzliche Methode der Planung ist das sogenannte *least commitment planning*.

- Vorwärts gerichtete Planung (progression/forward planning) - Die Suche beginnt mit dem Ausgangszustand
- Rückwärtsgerichtete Planung (regression/backward planning) - Die Suche beginnt mit dem Zielzustand
- least commitment planning - Formt die Suche im Zustandsgraphen um in eine Suche im Planungsraum, dem Raum, der erzeugt wird durch alle möglichen Pläne. Es ermöglicht somit, teilweise geordnete Pläne zu finden.

Dabei gibt es Planungsverfahren, die je nach Problem mehr oder weniger effizient, korrekt und vollständig arbeiten.

Graphplan

Graphplan ist ein 1995 von Avrim Blum und Merrick Furst entwickeltes Planungssystem, welches eine Problembeschreibung in STRIPS als Eingabe bekommt und einen teilweise geordneten Plan (*partial order plan*), so einer existiert, zurückgibt. Im Gegensatz zu den bisher beschriebenen Suchverfahren konstruiert Graphplan nicht den Suchbaum im Zustandsgraphen der Problembeschreibung, sondern einen, auf wechselnden Schichten von Aktionen und Bedingungen basierenden Flussgraphen. In diesem Graphen können Pläne mittels „*backward chaining*“ gefunden werden, also durch eine Suche nach dem Ausgangszustand beginnend mit dem Ziel.

Der Graph wird dabei während der Suche aufgebaut und erweitert. Somit handelt es sich also um ein auf iterativer Tiefensuche basierendes Verfahren. Der Graph setzt sich, wie bereits erwähnt, aus abwechselnden Schichten von

erfüllten Bedingungen und Aktionen zusammen. Zunächst wird der initiale Zustand als erste Ebene des Graphen hergenommen. Auf dieser Basis werden nun alle anwendbaren Aktionen konstruiert und in der zweiten Ebene gespeichert. Neben den in STRIPS beschriebenen Aktionen werden für dieses Planungsverfahren auch Aktionen benötigt, die keine Veränderung am gegebenen Zustand vornehmen. Diese „No-op"-Aktionen übertragen lediglich ein als „Wahr" gegebenes Prädikat in die nächste Bedingungsebene. Zusätzlich werden jene Aktionen markiert, die mit den gegebenen erfüllten Bedingungen nicht in beliebiger Reihenfolge ausgeführt werden können. Diese Aktionen schliessen sich gegenseitig aus, sind also „mutual exclusiv" - kurz „mutex". Aus allen Aktionen der zweiten Ebene wird nun die nächste Bedingungsebene konstruiert. Hierzu werden alle in der Aktionsbeschreibung vermerkten „Add-Effects" in die dritte Ebene aufgenommen. In dieser werden nun jene Prädikate als „mutex" markiert, welche sich zu diesem Zeitpunkt nicht gemeinsam in einem Pfad erfüllen lassen. Nur wenn es einen Pfad gibt, der beide Prädikate erfüllen kann, sind sie nicht „mutex".

Die Information über sich gegenseitig ausschliessende Aktionen und Prädikate erleichtert die Suche nach einem das Problem lösenden Plan. Ebenen, in denen zwei beliebige, im Zielzustand enthaltene Prädikate als „mutex" gekennzeichnet sind, können nicht Ausgangszustand der Suche sein, da sich sofort sagen lässt, dass kein Plan existiert, der alle Prädikate des Ziels erfüllt. Es muss also eine weitere Ebene hinzugefügt werden. Erst wenn keine der Ziele mehr „mutual exclusiv" sind, beginnt die Phase der eigentlichen Plansuche.

Diese beginnt - wie schon gesagt - mit der letzten konstruierten Ebene. Diese enthält alle Zielprädikate und es ist sichergestellt, dass es keinen Plan gibt, der mit weniger Ebenen auskommt. Leider lassen sich mit diesem Verfahren nur einfache Einschränkungen finden. Gibt es höherwertige Abhängigkeiten, wie etwa drei Prädikate, von denen zu einem gegebenen Zeitpunkt nur zwei beliebige gemeinsam erfüllt werden können, auf keinen Fall jedoch alle drei, so ist dies nicht im Graphen repräsentiert und kann somit die Plansuche nicht erleichtern. Bei der Suche selbst werden in der Aktionsebene vor der letzten Bedingungsebene mögliche Kombinationen von Aktionen berechnet, welche die benötigten Prädikate erfüllen. Aus einem

solchen Satz von Aktionen, die eine Ebene im „partial ordered Plan“ bilden, werden dann die Vorbedingungen aller Aktionen als Ziel der nächsten Ebene verwendet. Auf diese Weise wird mittels Tiefensuche ein Weg gesucht, der das Problem löst. Sollte es keinen Solchen geben, wird dem Graphen eine weitere Ebene von Aktionen und Bedingungen hinzugefügt und eine erneute Suche durchgeführt.

Während der Suche selbst werden die in der Graphkonstruktion gesammelten Informationen genutzt um mögliche, valide Aktionskombinationen in einer Ebene schneller zu finden.

geg.: Planungsproblem ($init$, $goal$, O)
$F0 := init$;
Erweiterung des Planungsgraphen in Schritt i:

1. füge jede Operatorinstanz $a = \sigma o$ mit $o \in O$ und $\sigma =$ Variablenbelegung als Aktionsknoten der Ebene i ein, falls $prec(a) \subset F_{i-1}$ und es gibt keine f_{i-1}^j, $f_{i-1}^k \in prec(a)$, die sich gegenseitig ausschliessen;

2. füge für jedes Faktum in F_{i-1} eine no-op Aktion, sowie die Vorbedingungskanten ein;

3. erstelle zu jeder Aktion a_i eine Liste von Aktionsknoten $\{b_i^j | j \in N, a_i$ und b_i^j schliessen sich gegenseitig aus$\}$;

4. erzeuge $F_i = \bigcup_{a \in A_i} add(a)$;

5. füge die Add- und Delete-Kanten ein;

6. markiere alle diejenigen Fakten f_i^j, f_i^k als sich gegenseitig ausschliessend, die nur durch sich gegenseitig ausschliessende Aktionen erzeugt werden.

Analyse des Planungsgraphen nach Schritt i:
$G := goal$;

`Search(G, i, Actions)`

selectedActions := $\emptyset$;
$G \notin F_i$ oder $\exists g1, g2 \in (G \cap F_i)$, die sich gegenseitig ausschlieen $\rightarrow$ `fail`;
$i = 0 \rightarrow$ `return(Actions)`;
`forall` $g_j \in G$ `do`
`choose` $a_i \in A_i$
`if` $g_j \in add(a)$ und ($a \in selectedActions$ oder es gibt kein $b \in selectedActions$, so dass sich a und b gegenseitig ausschliessen)
`then` $selectedActions := selectedActions \cup a_i$;
`else fail`;
`Search(` $\bigcup_{a \in selectedActions} prec(a)$, $i - 1$, $Actions \cup selectedActions$`)`

Graphplan ist besonders gut für Probleme geeignet, die entweder starke Parallellisierung zulassen oder bei denen sich häufig Abhängigkeiten erster Ordnung finden lassen. Dies gilt für die meisten „natürlichen“ Aufgabenstellungen und eignet sich hervorragend für das zentrale Bearbeiten von Multi-Agenten-Problemen.

1.3.7 Komplexere Planungsprobleme

Für „natürliche“ Problemstellungen, insbesondere bei der Planung von Handlungsabfolgen von Roboter-Agenten, ergeben sich Herausforderungen, die durch die klassischen Planungsmethoden nicht abgedeckt werden können. So kann in die Planung beispielsweise die Ausführungszeit von Aktionen einfliessen, was besonders für Multi-Roboter-Agenten-Systeme von Bedeutung ist, da es Probleme wie Synchronisation und Scheduling behandelt. Eine andere Forderung an die Planung ist ein eher Nutzenorientierter Ansatz, bei dem die Reihenfolge der Zielbearbeitung nach bestimmten, möglicherweise erlernten Kriterien erfolgen soll, um den Gesamtnutzen des Plans und nicht

dessen Länge zu optimieren. Weitere Beispiele wäre Planung mit Unsicherheit auf Grund dynamischer Umgebungen, unkoordinierte Multi-Agenten-Systeme oder nicht-deterministische Aktionen des Agenten.

1.3.8 Verteilte Planung

Von besonderem Interesse für die Forschung ist es, die zur Lösung eines Problems erforderliche künstliche Intelligenz auf mehrere Agenten aufzuteilen. Dabei unterscheidet man vier verschiedene Situationen der Verteilung geplanter Handlungsabläufe:

- Verteilte zentrale Planung - Verteilung der Planungsalgorithmen auf mehrere parallele Recheneinheiten.
- Zentrale Multi-Agenten Planung - Ein Planungssystem ermittelt einen Plan, dessen Ausführung auf mehrere Agenten aufgeteilt wird.
- Verteilte, kooperative Multi-Agenten Planung - Jeder Agent plant für sich selbst. Die Pläne werden anschliessend durch Koordination aufeinander abgestimmt (plan merging).
- Verteilte, nicht kooperative Multi-Agenten Planung - Jeder Agent hat seine eigenen Ziele und erzeugt einen Plan für sich selbst. Koordination zwischen den Agenten dient der Lösung von Ressourcenkonflikten.

Verteilte zentrale Planung ist ein Problem der Parallelisierung von Algorithmen, daher wird hier nicht weiter auf diesen Punkt eingegangen.

Zentrale Multi-Agenten Planung bedeutet, dass eine zentrale Instanz über das gesamte Wissen verfügt und mit dessen Hilfe einen Plan erzeugt, der einer Mehrzahl von Agenten verschiedene Aufgaben zuweist. Gerade für Probleme, deren Bearbeitung die Teilnahme mehrerer Agenten erfordert, eignet sich die Nutzung eines „*partial order planners*“, wie etwa Graphplan, da oftmals mehrere Agenten gleichzeitig und unabhängig von einander ihren Teil der Aufgabe bearbeiten können.

Der Vorteil der zentralen Planberechnung ist, dass die zentral planende Instanz über das vollständige Wissen verfügt und nicht nur Abhängigkeiten zwischen einzelnen Aktionen - auch von mehreren Agenten - zur Planungszeit mit einbeziehen kann, sondern auch die Verteilung der Aufgaben auf die Agenten zeitoptimal durchführt, also die Gesamtlänge des Plans minimiert - sofern der verwendete Planer optimale Pläne erzeugt.

Verteilte Multi-Agenten Planung bedeutet, dass jeder Agent seinen eigenen Plan anhand des ihm zur Verfügung stehenden Wissens erzeugt. Die auf diese Weise generierten Pläne sind Teilpläne des Gesamtsystems, die mittels Koordination zu einem globalen Plan zusammengerechnet werden müssen. Auf diese Weise erzeugte Pläne sind nicht optimal, obwohl die Agenten durch Absprachen Aufwand einsparen können.

Der grösste Vorteil dieses Systems von Teilplanung und Koordination ist die starke Reduktion des Suchraums. Gerade bei Multi-Agenten-Systemen mit verschiedenen Expertisen reduziert sich die Planung auf geringe Abschnitte des Gesamtplans.

Verteilte Planung in nicht Kooperativen Multi-Agenten-Systemen, in denen die einzelnen Agenten eigene Ziele verfolgen, die möglicherweise sogar einander widersprechen, ist die Königsdisziplin. Koordination wird in einem solchen System schwieriger, da sich diese nicht mehr zur Verbesserung des Gesamtnutzens einsetzen lässt. Es erfordert verschiedene Prinzipien der Verhandlungs- und Kooperationsstrategien. In einem solchen System spielen auch soziale Strukturen und Regeln eine bedeutende Rolle.

1.4 Verteilte Multi-Agenten Planung

In diesem Abschnitt werde ich einige Grundlagen der Koordination bei kooperativen Multi-Agenten-Systemen erläutern und anschliessend einige Ansätze vorstellen.

1.4.1 Grundlagen

Betrachtet man Multi-Agenten-Systeme, insbesondere kooperative, stellt sich die Frage, wieso man für diese überhaupt eigene Planungsverfahren benötigt und wieso diese sinnvoll sein sollen, obwohl doch keine optimalen Pläne gefunden werden können. Im Gegensatz zu einem zentral arbeitendem Planer, dem vollständige Informationen zur Verfügung stehen müssen, reichen verteilten Planungssystemen Ausschnitte der Gesamtinformationen. Der zentrale Planer muss nicht nur den relevanten Weltzustand, welcher den Zustand aller in die Planung einbezogenen Agenten einschliesst, kennen, sondern auch alle Informationen über die individuellen Fähigkeiten der Agenten, also die Aktionsbeschreibungen aller Agenten. Stehen diese Informationen zur Verfügung, kann ein Standard-Planungsvefahren daraus einen validen und optimalen Plan erzeugen und allen Agenten zur Ausführung mitteilen.

Wie schon gesagt, ist es jedoch nicht immer möglich, und oftmals auch nicht erwünscht oder zweckmässig, alle diese Informationen an einem Ort zusammen zu bringen. So könnten Agenten auf Grund eigener Interessen Informationen zurück halten. Insbesondere wird dies ersichtlich, wenn man berücksichtigt, dass Agenten nicht nur Software sondern auch Menschen sein können, die sich unter Umständen nicht einfach in einen zentralen Plan einbinden lassen.

Ein weiterer Grund für verteilte Planung ist Ausfallsicherheit. Ist eine zentral planende Einheit ausgefallen steht das Gesamtsystem still. Fällt in einem verteilt planenden System ein Agent aus, kann das gestellte Problem normalerweise immer noch gelöst werden - sofern keine notwendige Fähigkeit aus dem System genommen wurde. Insbesondere der Ausfall eines Agenten, dessen Typ mehrfach in einem System vorhanden ist, beeinträchtigt das System nur geringfügig.

Planung alleine ist also nicht aussreichend, da weder Vollständigkeit noch Korrektheit des Gesamtplans garantiert werden können. Um Abhängigkeiten in den Plänen mehrerer Agenten finden und einbeziehen zu können, müssen Agenten Informationen über ihre Pläne kommunizieren: sie müssen sich koordinieren. Eine Definition der Koordination nach Jennings ist:

> Participation in any social situation should be both simultaneously constraining, in that agents must make a contribution to it, and yet enriching, in that participation provides resources and opportunities which would otherwise be unavailable. Coordination, the process by which an agent reasons about its local actions and the (anticipated) actions of other agents to try to ensure the community acts in a coherent manner, is the key to achieving this objective.

In der Multi-Agenten-Planung ist es also die Koordination, die sicherstellt, dass trotz eigenständiger Planung und der trotz der Verfolgung eigener Interessen eines jeden Agenten die globalen Ziele des Gesamtsystems erfüllt werden können. Gleichzeitig ermöglicht die Koordination durch Kooperation sogar die Qualität des Gesamtplans zu verbessern.

Es gibt fünf Problemen, die eine Koordination in Multi-Agenten-Systemen erfordern:

1. Verhindern von Anarchie und Chaos - Agenten haben nur eine eingeschränkte, lokale Sicht auf das Gesamtproblem, können daher nicht erkennen, wenn gegenseitige Behinderung droht.

2. Effizienz - Agenten sind autonom und können ihre Aufgaben selbstständig bearbeiten. Kooperieren sie, kann sich die Gesamt-Effizienz unter Umständen deutlich erhöhen.

3. Globale Bedingungen erfüllen - Gibt es beispielsweise ein globales Budget müssen sich die Agenten absprechen um es nicht zu überschreiten.

4. Verteilte Informationen, Expertisen oder Ressourcen - Sind Probleme nicht von einem einzelnen Agenten bearbeitbar, müssen sich mehrere Agenten auf die gemeinsame Lösung des Problems einigen.

5. Abhängigkeiten zwischen Aktionen der Agenten - Die Agenten müssen sich über die Nutzung von Ressourcen einigen.

Für viele Koordinations-Probleme lassen sich drei immer wieder auftretende Aufgaben finden:

- Koordinierte Zielauswahl und -zerlegung,
- Koordinierte Ressourcen-Verteilung,
- Koordinierte Sequenzierung und Synchronisierung.

Durfee stellte zu Koordinationsansätzen fest:

> [...] various coordination strategies for computational agents have emerged over the years. It does not seem possible, however, to devise a coordination strategy that works well under all circumstances; if such a strategy existed, human societies would substitute it for the myriad constructs employed today such as corporations, governments, markets, teams, committees, professional societies, and mailing groups. Whatever strategy we adopt, certain situations can stress it to the breaking point.

Er identifiziert drei Kategorien von Bedingungen, die die Effizienz eines Koordinationsverfahren beeinflussen: *agent population*, *task environment* und *solution properties*. Die drei offensichtlichsten Eigenschaften jeder Kategorie sind:

agent population

- Quantity: Die Anzahl der Agenten.
- Heterogeneity: hierunter fallen z.B. unterschiedliche Fähigkeiten, interne Architekturen und Kommunikationsprookolle der Agenten.
- Complexity: bezeichnet hier den Grad der Vorhersagbarkeit des Verhaltens der Agenten.

task environment

- Degree of interaction: Beschreibt den Grad der Konkurrenz um gemeinsame Ressourcen. Beispielsweise ist eine Ressource, die den exklusiven Zugriff auf einen *blackboard datastore* eine Angelegenheit einer grossen Gruppe von Agenten, wohingegen etwa Kollisionsvermeidung nur jeweils eine kleine Gruppe betrifft.

- Dynamics: Ein Mass für die Geschwindigkeit mit der sich die Umgebung verändert; typischerweise abhängig von Ereignissen, die von den Agenten nicht beeinflussbar sind.

- Distributivity: Aufgaben können verteilt oder zentral behandelt werden.

solution properties

- Quality: Die Qualität einer Lösung kann beurteilt werden, anhand der Leistung bei der Koordination interagierender Agenten (z.B. fast optimal oder halbwegs akzeptabel) oder der Effizienz der Nutzung der Ressourcen der Agenten.

- Robustness: Ein Mass für die Robustheit eines Planes gegenüber Veränderungen der Umgebung; oder: wieviel Veränderung verträgt eine Planung bevor sie invalide wird.

- Overhead limitations: z.B. aufgrund begrenzter Kommunikations-Bandbreite.

De Weertd verallgemeinert bestehende Ansätze zu 6 Schritten in Koordinierungsproblemen:

1. *task refinement*: Brich die globalen Zielsetzungen oder Aufgaben soweit herunter, bis nur noch solche (Teil-)Aufgaben übrig bleiben, die einzelnen Agenten zugewiesen werden können.

2. *task allocation*: Ordne die so erhaltenen (Teil- und Unter-)Aufgaben eindeutig den Agenten zu.

3. *coordination before planning*: Definiere im Vorfeld Regeln und Ausschlüsse so, dass die Agenten keine widersprüchlichen Planungen produzieren.

4. *individual planning*: jeder Agent berechnet einen Plan unter Berücksichtigung seiner individuellen Ziele.

5. *coordination after planning*: Führe die Einzelplanungen der Agenten zu einem Globalplan zusammen.

6. *plan execution*: Die Ausführung der Teilpläne ergeben gemeinsam die Lösung des Problems.

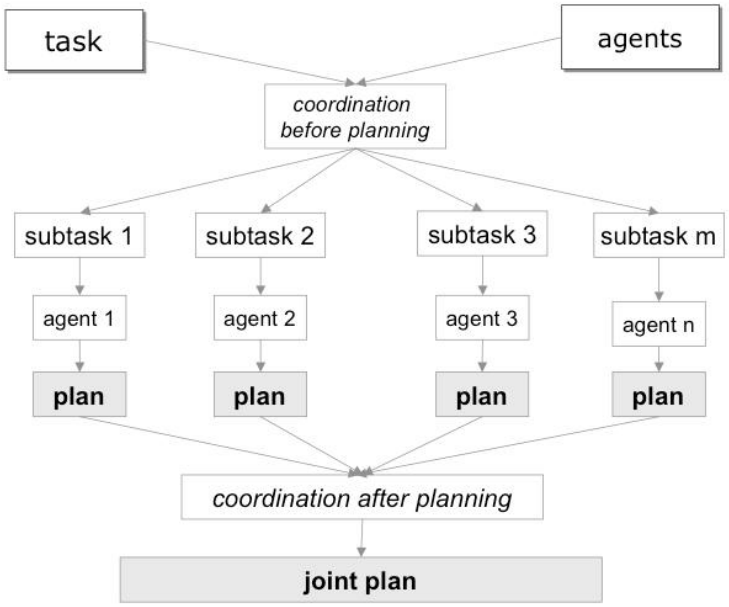

Abbildung 1.16: Prinzip der Koordination in einem MAS

Zu jedem dieser Schritte gibt es verschiedene Ansätze, die für bestimmte Szenarien mehr oder weniger gut geeignet sind. Die Koordinations-Phase kann grundsätzlich beliebig oft wiederholt werden, um immer bessere Gesamtpläne zu erhalten. In dieser Phase werden die Pläne - oder vereinfachte relevante Informationen über diese Pläne - ausgetauscht. Jeder Agent kann nun diese Pläne „*mergen*“, also zusammenführen, und versuchen, sie auf diese Weise zu verbessern, also Konflikte aufgrund knapper Ressourcen zu beseitigen und, falls erforderlich, Handlungen zu synchronisieren. Verändert ein Agent seinen Plan und zusätzlich den Plan eines anderen und kommt er dabei zu dem Ergebnis, dass der Nutzen der so entstandenen Pläne höher ist als der

Nutzen der Pläne vor der Veränderung, so schlägt der Agent seinem Kollegen diese Änderung vor. Dieser kann seinerseits den Vorschlag akzeptieren oder selbst einen Vorschlag abgeben. In mehreren Schritten wird auf diese Weise ein verbesserter Plan erzeugt. Auch wenn es nach mehreren Wiederholungen vorkommen kann, dass sich die Pläne auf diese Weise nicht mehr weiter verbessern lassen, ist nicht zwangsläufig ein optimaler Plan gefunden worden. Wahrscheinlicher ist, dass ein lokales Maximum erreicht wurde.

1.5 Roboter-Agenten

In diesem Abschnitt werde ich die Arbeit mit Roboter-Agenten genauer erläutern.

1.5.1 Grundlagen

Ihre Fähigkeit, sich im Raum zu bewegen, ermöglicht es, Agenten zur Lösung natürlicher Probleme einzusetzen. Dabei entstehen allerdings häufig neue Probleme, die zwar nur indirekt mit der Planung zusammenhängen, für das Funktionieren des Agenten als Teil eines Problemlösungssystems allerdings von entscheidender Bedeutung sind. Die grössten Probleme mobiler Agenten sind:

- Repräsentation der Umgebung
- Dynamik der Umgebung
- Bestimmung der eigenen Position

Diese Punkte sind jeweils eigene Forschungsbereiche und zu jedem gibt es mehrere Ansätze. Ich werde im Folgenden einige dieser Ansätze vorstellen.

1.5.2 Technische Voraussetzungen

Mobile Agenten stellen hohe Anforderungen an ihre technische Umsetzung, insbesondere, wenn sie im Sinne ihrer Definition autonom agieren sollen.

Der Agent benötigt neben der obligatorischen Ausstattung eines zentralprozessorgesteuerten Roboters Möglichkeiten der Interaktion mit seiner Umgebung, also Sensoren und Aktoren. Als Sensoren kommen verschiedene Formen von Abstandsmessern, Licht- oder Wärmefühlern, Kameras und Funk in Frage. Natürlich lässt sich diese Liste noch erweitern. Aktoren könnten neben dem Fortbewegungsmechanismus beispielsweise Greifarme oder dem Einsatz zweckentsprechende Werkzeuge sein. Für Multi-Agenten-Systeme mit speziallisierten Expertisen könnte es zum Beispiel einen Typ Agent geben, der neben Rädern zur Fortbewegung noch einen Feuerlöscher zur Brandbekämpfung hat und einen weiteren, der einen Greifarm zum Entfernen von Trümmern besitzt.

1.5.3 Repräsentation der Umgebung

Wenn im Zusammenhang mit Planung von „Umgebung“ gesprochen wird, ist darunter die Beschreibung der relevanten Eigenschaften der Umgebung des Agenten zu verstehen. Bei mobilen Agenten ist der Begriff „Umgebung“ tatsächlich wörtlich zu nehmen. Der Agent bewegt sich, mehr oder weniger frei, im realen Raum. Um in diesem zielgerichtet handeln zu können, muss dem Agenten eine vom Anwendungsfall abhängige, mehr oder weniger abstrahierte Repräsentation dieses Raumes zur Verfügung stehen. Diese sollte so detailliert umgesetzt sein, dass benötigte Informationen extrahiert werden können. So muss diese Abbildung des Raumes beispielsweise Wegplanung ermöglichen. Diese Repräsentationen der Umgebung werden als Karten bezeichnet.

Es gibt im Grunde zwei Arten von Karten. Die erste Art ist die Rasterkarte. Bei Rasterkarten wird die Welt auf eine in der Technik üblichen Weise digitalisiert, indem die analogen Messdaten in digitale Informationen umgerechnet werden. Dadurch entstehen Karten die aus Feldern - im 2-dimensionalen Umgebungen - bestehen. Diese Felder enthalten Informationen über die Passierbarkeit oder Unpassierbarkeit der Welt an dieser Stelle. Grundsätzlich können verschiedene Informationen in einer Rasterkarte gespeichert werden. Bei zunächst unbekannten Umgebungen ist es üblich, dass die Rasterfelder

zunächst den Wert „unbekannt“ haben und bei Erreichen neuen Wissens auf „blockiert“ oder „frei“ gesetzt werden. Eine andere Möglichkeit ist es, zunächst jedes Feld als frei anzunehmen und nach Erhalt entsprechender Informationen Felder auf „blockiert“ zu setzen. Durchaus üblich ist auch die Speicherung von Höheninformationen in den Feldern.

Die zweite Art ist die topologische Karte. Diese beinhaltet Informationen über die Welt in Form von Landmarken und identifizierbaren Geländeabschnitten. Üblicherweise werden Strassennetze topologisch, in Form eines Graphen gespeichert, bei dem Abzweigungen oder Kreuzungen die Knoten und die Strassen die Kanten sind. Der Vorteil dieses Verfahrens ist die erhebliche Reduzierung des Speicheraufwands und der zur Wegplanung benötigten Informationen.

1.5.4 Dynamik der Umgebung

Wenn man die Dynamik der Umwelt eines Agenten betrachten will, muss man vier grundlegende Prinzipien der Veränderbarkeit unterscheiden:

- Veränderung der Umgebung durch eigene Regeln
- Veränderung der Umgebung durch Aktionen des Agenten
- Veränderung der Umgebung durch andere Agenten
- Veränderung durch Äussere Einwirkung

Während die dynamische Veränderung der Umwelt sich, innerhalb gewisser Grenzen, in der Repräsentation der Umgebung wiedergeben lässt und Veränderungen durch andere Agenten sich mittels Kommunikation zwischen diesen einplanen lassen, sind die Veränderungen durch Dritte nicht vorhersehbar. Der Agent muss dennoch in der Lage sein auf solche Veränderungen reagieren zu können.

In einer dynamischen Umgebung verändern sich Eigenschaften der Umgebung ohne Einwirkung des Agenten. Oft lassen sich diese Veränderungen in Regeln fassen und zumindest teilweise vorhersagen. Solange die Veränderung

Regeln unterliegt oder aber der Zustand bei der Nutzung zum Planungszeitpunkt nicht feststeht, jedoch als ungenaues Wissen verfügbar ist, lassen sich nach wie vor valide Pläne erzeugen. Als ungenaues Wissen bezeichnet man Informationen, die dem Agenten zur Verfügung stehen, die aber nur eine Vermutung über den Weltzustand darstellen. Beispielsweise sind auf einer topologischen Karte alle Strassen bekannt, es kann aber nur vermutet werden, dass sie auch alle befahrbar sind. Erst durch Verifikation des Zustands lässt sich das Wissen festigen oder widerlegen. Ungenaues Wissen wäre in dem Beispiel der Strassenkarte auch die typische Stauwahrscheinlichkeit einer Strasse. Der Agent kann mit diesem ungenauen Wissen immer noch planen, mit Hilfe von probabilistischen Plänen kann er sogar einen Plan erzeugen, der alle Möglichkeiten beinhaltet.

Ein Sonderfall ist das Planen mit zunächst unbekanntem Wissen. Unbekanntes Wissen fällt, ungeachtet dessen, ob es dynamisch oder statisch ist, in die selbe Kategorie wie äussere Einwirkung. Es lässt sich nicht in einen Plan einbinden. Somit kann es keinen Plan geben, der unbekanntes Wissen einschliesst. Sobald jedoch Mutmassungen über noch nicht verifiziertes Wissen angestellt werden können, ist es streng genommen nicht mehr unbekannt sondern schlicht ungenau.

1.5.5 Positionsbestimmung

Die Bestimmung der eigenen Position ist ein essenzieller Bestandteil der zielgerichteten Navigation. Erst wenn die eigene Position bekannt ist, lässt sich feststellen, wo alles andere ist.

Es gibt verschiedene Verfahren, die je nach Einsatzgebiet mehr oder weniger erfolgversprechend sind. So ist es beispielsweise möglich, einen Agenten mit einem GPS-Empfänger auszustatten und ihm so seine Position zukommen zu lassen. Da GPS für die zivile Nutzung nur bis auf ca. 15 Meter genau angegeben wird ist die Nutzung dieses Systems nur bei Agenten sinnvoll, denen eine unscharfe Positionsinformation ausreicht, beispielsweise für ein Fahrzeug in der Grösse eines normalen PKW, das sich im Gelände orientieren soll. Selbst wenn das System präzise Positionsbestimmung zuliesse, wäre

der Agent auf den ungehinderten Empfang der GPS-Daten von den Satelliten angewiesen, was zum Beispiel unter der Erdoberfläche, in Höhlen, Tunneln oder Rohrsystemen nicht gewährleistet ist.

Für solche Systeme, die im Grunde labyrinth-artige Probleme bilden, lassen sich besser „*occupancy grid*“ basierte Landmarken-Karten einsetzen. Hierzu wird jeder Knotenpunkt des Labyrinthes, welches in Form eines Graphen dargestellt wird, eindeutig identifiziert. Zwischen den Knotenpunkten kann dann ohne genaue Kenntnis der Position hin und her navigiert werden, da immer sichergestellt ist, dass durch einfaches Folgen eines Ganges zwangsläufig der nächste Knoten erreicht wird.

Im Falle der Navigation in weitestgehend unstrukturierten Umgebungen, wird ohne vorhandene Orientierungshilfen die Positionsbestimmung zunehmend komplex und ungenau, sofern nicht ein äusseres Orientierungssystem wie GPS vorhanden ist. Die einfachste Form ist in einem solchen Fall die Verwendung von Odometrie, also der Positionsbestimmung anhand der eigenen Bewegung. Ist der Agent beispielsweise mit Rädern ausgestattet, lässt sich die Position relativ zu einer vorher bereits bekannten Position bestimmen, indem man die Radumdrehungen auf beiden Seiten zählt. In regelmässigen, kurzen Zeitabschnitten lässt sich daraus die neue Position berechnen. Genauer handelt es sich um die Bestimmung der relativen Veränderung der Position aus der gemessenen Bewegung des Roboters. Aus den Radumdrehungen in bestimmten Zeitintervallen kann man unter Berücksichtigung des Umfangs der Räder die von jedem Rad zurückgelegten Strecken D_L und D_R berechnen. Ist die Strecke auf einer Seite länger als auf der Anderen, so ist der Roboter in dem gegebenen Zeitintervall offensichtlich eine Kurve gefahren. Sind die Zeitintervalle kurz genug gewählt, so lässt sich die neue Orientierung des Roboters über die Formel $O_{T+1} = O_T + (D_R - D_L)/W$ berechnen. Die neuen kartesischen Koordinaten des Roboters berechnen sich dann als:

$$X_{T+1} = X_T + D_{T,T+1} * cos(O_{T+1})$$

$$Y_{T+1} = Y_T + D_{T,T+1} * sin(O_{T+1})$$

Die auf diese Weise berechnete Position ist jedoch nur eine Abschätzung

und benötigt Methoden der Synchronisierung mit der verwendeten Karte. Es müssen mehrere Fehlerquellen beachtet werden. Die deutlichsten Fehler bei dieser Form der Positionsbestimmung entstehen durch:

- Ungenauigkeiten in der Erfassung der Umdrehungszahl der Räder
- Ungenauigkeiten im Radius der Räder
- Ungenauigkeit im Abstand der Räder
- Durchdrehende Räder aufgrund mangelnder Reibung
- Unebenes Gelände

Selbst wenn die Fehler durch technische Mittel reduziert werden können, werden die daraus resultierenden Abweichungen sich über die Zeit immer aufsummieren.

Der Agent muss deshalb - zumindest von Zeit zu Zeit - die Möglichkeit haben, seine eigene Position zu aktualisieren und somit sein eigenes Koordinatensystem mit dem „globalen“ abzugleichen. Zu diesem Zweck nutzt man Landmarken, die der Agent erkennen kann und deren Position eindeutig bekannt ist. Das Erkennen und Setzen von Landmarken ist ein eigenes Forschungsthema, dessen Kern ich hier nur kurz erläutern werde.

Um Landmarken in einer freien Umgebung setzen zu können muss der Agent zunächst mit der Fähigkeit ausgestattet sein, charakteristische Merkmale seiner Umgebung zu erkennen und eindeutig bestimmen zu können. Hierzu können entweder Prinzipien der Objekterkennung oder der Strukturerkennung eingesetzt werden.

Im Falle der Objekterkennung werden Landschaftsmarken wie etwa eindeutig erkennbare Bäume, Gebäude oder Schilder als Orientierungshilfe herangezogen. Dies eignet sich insbesondere für Agenten mit optischer Sensorik, die solche Objekte auf eine gewisse Distanz erkennen können. Bereits mit zwei sichtbaren, erkannten Objekten lässt sich die eigene Position sehr genau bestimmen. Für Agenten ohne die Möglichkeit der Odometrie ist die optische Erfassung von charakteristischen Objekten auch die beste, wenn nicht die einzige Methode, die eigene Bewegung zu ermitteln. Können ausreichend

viele solcher Landmarken auf möglichst grosse Entfernung erkannt werden, lässt sich auch das Problem freier Bewegung auf ein Graphenproblem abbilden. Die Navigation erfolgt dann entlang der Orientierungshilfen, während Positionen im freien Raum relativ zu den Landmarken angegeben werden können. Der grösste Vorteil eines solchen Systems ist, dass auch eine spontane Positionierung im Raum mittels bekannter Landmarken möglich ist. Dies ermöglicht einem Agenten zum Beispiel, seine Position zu bestimmen, wenn er an einem beliebigen Ort ausgesetzt wird und dort erst seine automatischen Handlungen aufnimmt.

Eine solche freie Navigation anhand von Gebäuden und anderen Landschaftsmerkmalen ist das höchste Ziel der Roboternavigation. Zur Zeit scheitert eine allgemein nutzbare Landmarken-basierte Navigation noch an dem enormen Rechenaufwand, den die bildbasierte Objekterkennung erfordert. Ein vereinfachtes Konzept beruht auf der manuellen Platzierung von Leuchtfeuern (engl. *beacon*), die für einen Agenten leicht erkennbar sind. Da dies jedoch einen Eingriff in das Einsatzgebiet des Agenten erfordert, ist es praktisch nur in Einzelfällen nutzbar. Ein Beispiel hierfür sind Funkstationen, die der Agent anpeilen kann oder Markierungen an Wänden und Boden, anhand derer sich der Agent im Inneren eines Gebäudes zurecht finden kann.

Die Positionsbestimmung mittels Analyse der Umgebungskartenstruktur ist hauptsächlich für den Einsatz im Inneren eines Gebäudes gedacht. Der Agent vermisst seine Umgebung und erzeugt so einen „Fingerabdruck“ eines Raumes, einer Flurecke oder eines Durchgangs. Sind diese Fingerabdrücke eindeutig genug, um bei erneutem Vermessen der Umgebung wiedererkannt zu werden, lassen sich diese als Landmarken einsetzen. Im einfachsten Fall lassen sich z.B. Raumecken nutzen. Reicht die geschätzte Position aus, um feststellen zu können, in welche Ecke der Agent gerade steuert, kann er den Fehler bei Erreichen einer bestimmten Position in der Ecke herausrechnen. Steuert er regelmässig bekannte Positionen an, lässt sich so - auch dynamisch - eine recht genaue Karte erzeugen.

1.5.6 Mobile Multi-Agenten-Systeme

Wenn die Agenten eines Multi-Agenten-Systems mobil sind, sich also frei im Raum bewegen können, entstehen neue, koordinationsbedürftige Abhängigkeiten, da nun auch der Raum eine exklusive Ressource darstellt. Die Auswirkungen dieser neuen Ressource - oder besser der Mangel an ihr - können wir ständig auf unseren Autobahnen beobachten. Um solche Konflikte zu vermeiden, müssen die Agenten auch den Platz, den sie zu einem bestimmten Zeitpunkt einnehmen werden, in ihren Plan und den daraus erzeugten globalen Gemeinschaftsplan einbeziehen. Eine effektive Planung der Besetzungsrechte bestimmter Gebiete ist jedoch nur mit einer gleichzeitigen Planung der Zeit möglich, da ansonsten Konflikte gefunden werden, die durch die zeitverschobene Nutzung der Ressource tatsächlich gar nicht auftreten. Obgleich dies auch für andere Ressourcen gilt, ist deren Zuteilung oft reaktiv möglich, also der Nächste erhält die Ressource sobald ein Anderer sie freigibt. Bei Problemen im 2- oder 3-dimensionalen Raum lässt sich jedoch im Grunde immer ein Alternativweg finden, der zwar möglicherweise einen Umweg darstellt, dafür aber einen sich abzeichnenden Konflikt beseitigt. Es bedarf also einer Synchronisation der Pläne und deren Ausführung, um erkennen zu können, ob die Ressource Raum zur selben Zeit oder nacheinander genutzt werden soll oder ob es effizienter ist, einen alternativen Weg zu benutzen.

Viele Probleme der gleichzeitigen Navigation lassen sich durch die Einführung von Preplanungsregeln oder Ausführungszeitregeln vermeiden. Im ersten Fall können, besonders bei Strassenkarten und ähnlichen Graphenproblemen, Verkehrsregeln, wie etwa ein Rechtsfahrgebot, das Entstehen von Konflikten unterbinden. Im zweiten Fall werden dem Agenten gewisse Freiheiten bei der Durchführung des Plans gelassen, die es ihm ermöglichen, auf auftretende Situationen angemessen zu reagieren. So kann beispielsweise „Rechts vor Links“ oder auch das Stoppen an einer Ampel situativ erfolgen, ohne vorher explizit in einem dynamischen Plan festgelegt zu sein. Kollisionsvermeidung ist ebenfalls ein starker Kandidat für reaktives statt geplantes Handeln.

Kapitel 2

Umsetzung

2.1 Rescue Szenario

2.1.1 RoboCupRescue

Das dieser Arbeit zugrunde liegende Szenario ist eine Variante des Rescue Szenarios, welches die RoboCupRescue League verwendet. Diese hat es sich zur Aufgabe gemacht, Forschung und Entwicklung in verschiedenen Bereichen der Rettung in Katastrophensituationen zu fördern. Katastrophenmanagement, besonders in grossen Städten, ist eines der kritischsten organisatorischen Probleme der modernen Gesellschaft. Es muss mit einer grossen Zahl heterogener Agenten in einer feindseligen Umgebung gearbeitet werden. RoboCupRescue unterstützt Forschung in verschiedenen Bereichen dieses Szenarios durch Austausch und Vergleich verschiedener Projekte. Diese beinhalten Multi-Agenten Koordination, physische Roboter zur Suche und Rettung, Informationsinfrastrukturen, personal digital assistants, standardisierte Simulatoren, entscheidungsunterstützende Systeme, Evaluationssysteme zur Bewertung von Strategien und Roboter-Systeme, die in alle Bereiche eingebunden sind.

Die Simulations-Liga, die Teil von RoboCupRescue ist, beschäftigt sich in erster Linie mit dem Verhalten der Agenten, untersucht also beispielsweise Multi-Agenten-Planung, Echtzeitplanung, Umgang mit Heterogenität der Roboter und mit der Robustheit der Pläne. Auch die Entwicklung und

Verbesserung der Simulation der einzelnen Komponenten des Szenarios, z.B. Brandausbreitung, gehört dieser Liga an. Grundlage hierfür ist die Simulation einer Stadt, die sich im Katastrophenzustand befindet. Die der Simulation zugrunde liegende Stadtkarte ist mitsamt Strassen und Häuserblocks bekannt; die Position sowie das Ausmass einzelner Probleme muss hingegen erkundet werden. Um die Katastrophen-Simulation nicht zu unübersichtlich werden zu lassen, beschränkt man sich auf drei verschiedene Problemstellungen: Brände, Eingeschlossene oder Verletzte und blockierte Strassen. Zur Lösung dieser Probleme gibt es drei verschieden spezialisierte Agententypen: Polizei, die Blockaden entfernt, Feuerwehr, die Brände löscht und Rettungsdienste, die Verletzten helfen. Jeder Gruppe von Agenten stehen ausserdem Komandozentralen zur Verfügung, die Informationen sammeln, Aufgaben verteilen und die Einsätze der verschiedenen Gruppen koordinieren.

Aus Sicht der künstlichen Intelligenz ergeben sich in diesem Szenario unter anderem folgende Betätigungsfelder:

- Rasche und gründliche Erkundung der Umgebung um die Probleme zu finden
- Bewertung der „Dringlichkeit“ und der Wertigkeit der Probleme
- (verteilte) Planung mit unsicherem Wissen
- Informations- und Aufgabenverteilung

Die zeitliche Beschränkung des Szenarios erfordert es, so viele der vorhandenen Probleme wie möglich zu finden, zu bewerten und zu lösen und das so schnell wie möglich. Anhand des unsicheren Wissens lassen sich nur in eingeschränktem Masse Pläne erzeugen. Das Hauptaugenmerk bei allen Lösungsansätzen liegt daher im Allgemeinen auf der Bewertung der einzelnen Probleme und auf möglichst effektiven Erkundungsstrategien.

Da es nicht Ziel dieses Szenarios ist, alle vorhandenen Probleme zu lösen und zu viele Details der Umgebung unbekannt oder ungewiss sind, wird im Allgemeinen keine vollständige Planung eingesetzt, sondern eine Aufgabenverteilung auf der Grundlage möglichst sorgfältiger Bewertung der Probleme,

gefolgt von einer lokalen Planung des mit der jeweiligen Aufgabe betrauten Agenten.

Es muss auch abgeschätzt werden ob die Bearbeitung bereits erkannter Probleme immer Vorrang vor weiterer Erkundung der Umgebung hat oder ob es zweckdienlicher ist, zunächst möglichst viele Probleme zu finden und diese erst anschliessend zu bearbeiten.

2.1.2 Aufgabenstellung

Die hier beschriebene Arbeit basiert auf einem ähnlichen Szenario. Im Gegensatz zu RoboCupRescue gibt es jedoch kein festes Strassennetz und auch alle anderen Informationen über die Umgebung sind zunächst unbekannt. Die beteiligten Agenten müssen ihre Umgebung also nicht nur nach Problemen durchsuchen, sondern auch die Struktur der Umgebung erkunden. Um dabei einen gezielten Informationsaustausch über die Umgebung zu ermöglichen, wird vorausgesetzt, dass charakteristische Merkmale der Umgebung, so genannte Landmarken, immer und eindeutig von jedem Agenten erkannt und identifiziert werden können. Landschafts- oder Objekterkennung sind nicht Bestandteil dieser Arbeit, daher werden die Landmarken als sich selbst identifizierende Objekte Teil der Umgebung sein. Es wird zwei Problemtypen geben: Brände und Verletzte, die jeweils an eine Landmarke gebunden sind. Entsprechend gib es zwei Agententypen: Brandbekämpfer und Sanitäter. Zusätzlich wird noch ein Agententyp hinzugenommen, der keine eigenen Fähigkeiten hat und nur zur Unterstützung bei der Erkundung dient.

Aufgabe der Agenten ist es, eine Anzahl von Bränden und Verletzten zu finden und darauf hin entsprechende Massnahmen zu ergreifen, wobei kein Zeitrahmen vorgegeben ist. Durch Planung soll eine möglichst effiziente Handlungsabfolge für alle Agenten gefunden werden. Die Grundidee verteilter Multi-Agenten-Systeme, insbesondere die Autonomie der Agenten bei der Planerzeugung, soll hierbei nicht in Frage gestellt werden. Daher entsteht der Einsatzplan im ersten Schritt verteilt; mittels Koordination werden die Teilpläne zusammengefürt und auf Korrektheit geprüft. Schliesslich soll der korrekte, globale Plan dann durch weitere Methoden der Koordination

verbessert werden.

Zur Umsetzung stehen Khepera-II-Roboter und die Simulation Webots zur Verfügung. Die Entwicklung erfolgt in C++, was eine direkte Umsetzung des Steuerprogramms auf den Roboter erschwert. Es ist aufgrund des Planungsaufwands, mit dem nicht nur offline - vor der Arbeit - sondern besonders online - während der Bearbeitung - zu rechnen ist, nicht anzunehmen, dass Speicher und Rechenleistung des Roboters ausreichen, um das Steuerprogramm direkt auf diesem ausführen zu können. Es wird daher in Erwägung gezogen, die Bewegungssteuerung der Roboter über eine Fernsteuerung erfolgen zu lassen; das eigentliche Steuerprogramm könnte dann auf einem aussreichend leistungsfähigen PC ablaufen. Eine solche Variante müsste allerdings auch berücksichtigen, dass nicht nur Steuerbefehle vom Rechner an die Roboter sondern umgekehrt auch Sensordaten der Roboter an den Rechner übermittelt werden müssen. Da die Entwicklung jedoch zunächst mit Webots erfolgen wird, kann dieses Problem zunächst ausser acht gelassen werden. Zusätzlich werden vorweg einige Vereinfachungen festgelegt:

- Batteriestand der Roboter wird nicht beachtet; in der Simulation ist er nicht begrenzt.
- Die Aktionen der Roboter sind abstrakt; es ist ausreichend, wenn sie am richtigen Ort mitteilen, dass sie eine Aktion ausführen. Aktionen benötigen keine Zeit.
- Die Aufgaben und Aktionen sind an die Landmarken gebunden; es ist ausreichend, wenn sich ein Agent an einer Landmarke befindet, um entsprechende Aktionen ausführen zu können

Diese Arbeit beschäftigt sich mit der Erzeugung gültiger, globaler Pläne. Zur Planung wird, wie bereits erläutert, das vollständige Wissen über die Umgebung benötigt. Es gibt zwar Verfahren, die mit vollständigem aber unsicherem Wissen Pläne erzeugen - *Bedingtes Planen* -; da unbekanntes Wissen aber nicht verfügbar ist, kann es auch nicht in einen Plan einbezogen werden. Durch die Verteilung des Wissens auf die Agenten steht jedoch sowieso nicht jedem Agenten das vollständige Wissen über das Gesamtsystem

zur Verfügung. Es gilt also zu untersuchen, wie mit unbekanntem Wissen umgegangen werden kann.

Das Ziel dieser Arbeit ist, an einem praktischen Beispiel verteilte Planung und zentrale Planung zu vergleichen. Gerade durch die zunächst unbekannte Umgebung stellen sich Fragen über Effizienz der verteilten Planung, Korrektheit, Vollständigkeit, Optimalität oder Robustheit. Ausserdem soll diese Arbeit als Grundlage für weitere Untersuchungen in diesem Bereich dienen.

2.2 Khepera

Abbildung 2.1: Khepera-II Roboter

In dieser Arbeit wird der Khepera-II Roboter als „körperliche“ Repräsentation der Agenten genutzt. Der Khepera ist ein mobiler Miniatur-Roboter der ursprünglich 1992 als Forschungs- und Unterichtswerkzeug im Rahmen eines *Swiss Research Priority Program* entworfen wurde. Die Funktionalität des Khepera ist dabei durchaus mit der grösserer Roboter vergleichbar. Er ermöglicht das Testen von, in Simulationen entworfenen, Algorithmen in der realen Welt. Unter anderem zählen Zielgerichtetes Planen, Hindernisvermeidung und Sensorverarbeitung zu den möglichen Anwendungsbereichen.

Ein entscheidender Vorteil des Khepera ist die hohe Modularität sowohl auf der Software- als auch der Hardware-Ebene. Es existiert eine Vielzahl von Erweiterungsmodulen, die dem Khepera neue Fähigkeiten verleihen und es somit ermöglichen, ihn in einem breiten Spektrum von Szenarien einzusetzen. Umfassende und effiziente Software-Bibliotheken für Steuerung, Sensorik und Analyse erleichtern die Entwicklung neuer Steuerungsalgorithmen.

ELEMENTS	TECHNICAL INFORMATION
Processor	Motorola 68331
RAM	256 Kbytes
ROM	256 or 512 Kbytes
Motion	2 DC motors with incremental encoder
Sensors	8 Infra-red proximity and light sensors
Power	Rechargeable NiCd Batteries or external
Autonomy	30 minutes (basic configuration with maximal activity)
Extension Bus	The robot can be expanded by modules added on the K-Extension bus
Diameter	55 mm
Height	30 mm
Weight	About 70 g

Tabelle 2.1: Technische Informationen Khepera-II

Der Khepera ist deshalb für Forschungszwecke so interessant, weil durch seine geringe Grösse Versuche in einem räumlich begrenzten Umfeld, beispielsweise auf einem Tisch, möglich sind. Zur Modellierung einer Umgebung können Gegenstände verwendet werden, die gerade zur Hand sind, was spontane Versuche ermöglicht und Änderungen an Szenarien erleichtert. Kheperas

sind zudem einfach transportierbar und am neuen Ort sofort einsatzbereit, was sie insbesondere für Vorführungen und Demonstrationen geeignet macht.

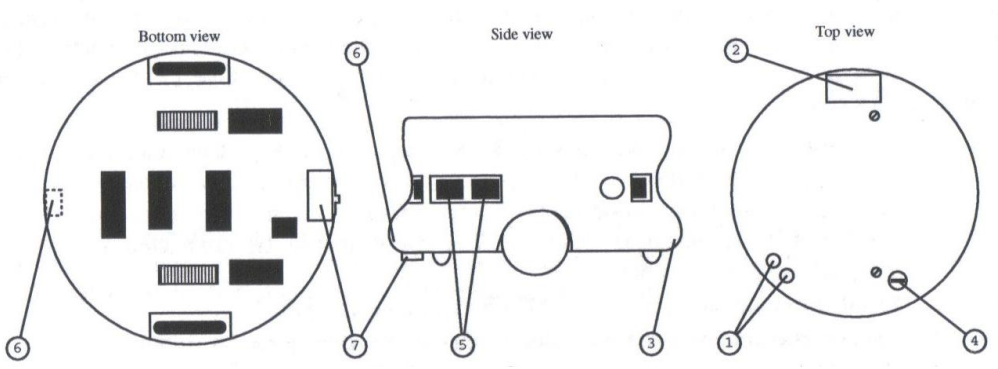

Abbildung 2.2: Overview Khepera-II: 1. LEDs 2. Serial line connector 3. Reset button 4. Encoding wheel to select running mode 5. Infra-Red proximity sensors 6. Battery charger connector 7. ON - OFF battery switch

Besonders Szenarien mit mehreren Khepera-Robotern, die beispielsweise ein gemeinsames Problem bearbeiten, sind ohne weiteres möglich. Sie werden durch die verhältnissmässig geringen Anschaffungskosten und die Einheitlichkeit der Roboter erleichtert und durch Module wie etwa den *radio-turret* unterstützt.

Die Programmierung der Steuerprogramme erfolgt entweder in C oder in der M68000 Assembler Sprache. Diese werden mit einem Cross Compiler übersetzt und dann in den RAM des Roboters geladen, wo der Roboter sie auslesen und ausführen kann. Für die einfache Nutzung der Hardware des Roboters steht eine vollständige API in beiden Sprachen zur Verfügung.

Einfacher ist die Verwendung von LabView oder anderer Werkzeuge (Matlab, Sysquake, Webots, YAKS) als Entwicklungsumgebung.

Der Khepera-II Roboter ist mit acht TCRT1000 infra-rot Sensoren ausgestattet deren Anordnung in der Abbildung 2.3 dargestellt sind. Diese Anordnung ergibt eine gute Sensorabdeckung des vorderen Aktionsbereichs. Die infra-rot Sensoren erfüllen zwei Aufgaben:

- Die Sensoren können Umgebungslicht messen. Dies ermöglicht beispielsweise Reaktionen auf Lichtquellen.

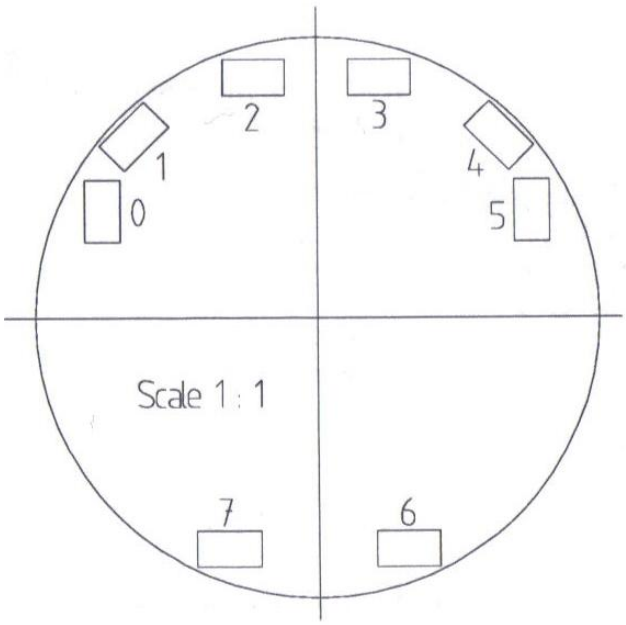

Abbildung 2.3: Anordnung der Infra-Rot Sensoren

- Die Sensoren können zur Entfernungsmessung von Hindernissen genutzt werden.

Für die Entfernungsmessung wird aktiv Licht ausgesendet und der Unterschied zwischen ausgesandter und empfangener Lichtintensität - um das Umgebungslicht korrigiert - gemessen. Die Sensoren haben dabei eine Reichweite von ungefähr 10 cm, wobei die Messgenauigkeit mit wachsendem Abstand abnimmt. Die Sensoren dienen daher in erster Linie der Erkennung von Hindernissen in der direkten Umgebung des Roboters. Sie eignen sich, unterstützt auch durch die Anordnung, ideal zur Kollisionsvermeidung beispielsweise durch den Braitenberg-Algorithmus.

2.3 Webots

Webots ist eine in Forschungseinrichtungen häufig genutzte kommerzielle Software, die das Modellieren, Programmieren und Simulieren von Robotern in einer beliebigen Umgebung ermöglicht. Integrierte Bibliotheken erleichtern

es dem Benutzer eine unter Webots entwickelte Roboter-Steuerung direkt auf den echten Roboter zu übertragen.

In Webots modellierte Roboter können beliebige Form und Ausstattung haben, durch ODE (*Open Dynamics Engine*) ist eine realistische Simulation der Auswirknugen physikalischer Gesetze auf das Robotermodell anwendbar. Eine auf VRML97 basierende Beschreibungssprache erlaubt auch die freie, hierarchische Definition der Umgebung. Jedes Objekt der Umgebung kann über eine Reihe von Eigenschaften, beispielsweise Form, Farbe, Textur oder Masse angepasst werden. Webots bringt zur Vereinfachung dieser Arbeiten eine Graphische Oberfläche mit, unter der auch die Simulation läuft. Viele der in VRML97 definierten Knoten existieren auch in Webots. Neben den primitiven geometrischen Formen `Box`, `Cylinder` und `Sphere` ist der Gruppierungsknoten `Group` und der Transformationsknoten `Transform` besonders wichtig. Des Weiteren enthält die Beschreibung einer Welt noch `Viewpoint`, `Shape`, `Material` und `Appearance` Knoten.

Webots kennt zusätzliche Knoten wie `Robot` und dessen Speziallisierung `DifferentialWheels`, `Solid`, eine Erweiterung von `Transform`, der für die Kollisionsberechnung benötigt wird und Sensor- und Aktor-Knoten wie: `Camera`, `TouchSensor` oder `Gripper`.

Beispiel Für das hier betrachtete Szenario wird eine Umgebung benötigt, die Hindernisse enthält und Platz für mehrere Khepera-II Roboter bietet. Der Khepera-II ist ein „Tischroboter“, daher wird als Untergrund eine Platte von 2*2 Meter gewählt. Hierzu wird in den hierarchischen Szenengraphen ein `Solid` Knoten eingefügt. Zur graphischen Darstellung wird diesem Knoten ein `Shape` Knoten angegliedert und, für die Kollisionserkennung, ein `bounding object` definiert. Beide entsprechen einer flachen Box mit 2*2 Meter Kantenlänge. Erweitert um einen `Appearance` Knoten mit einer Textur wird diese Bodenplatte von der Simulation wie in Abbildung 2.4 dargestellt.

Der Khepera-II ist jedoch nicht in der Lage, Vertiefungen in der Umgebung zu erkennen, also die Tischkante als solche wahrzunehmen. Daher benötigt das Szenario eine äussere Umrandung als Begrenzung. Diese kann beispielsweise aus vier weiteren `Solid`-Knoten bestehen, die jeweils so de-

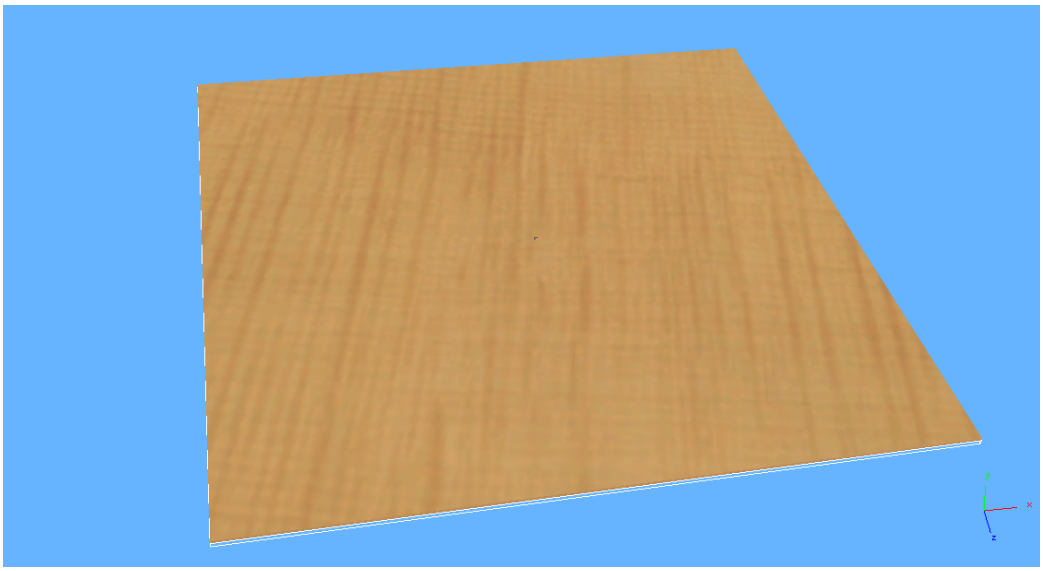

Abbildung 2.4: Webots: Bodenplatte

finiert sind, dass sie eine Seite begrenzen. `Solid` ist eine Erweiterung von `Transform` und enthält daher auch ein entsprechendes Feld mit dem die einzelnen Wandstücke an den Rand der Bodenplatte verschoben werden. Abbildung 2.5 zeigt die Bodenplatte mit äusserer Begrenzung.

Als nächstes wird ein Khepera-II-Roboter in dieser Umgebung platziert. Webots enthält Modell und Beschreibung des Khepera-II Roboters, der aus einer Liste verschiedener `DifferentialWheel`-Robotern ausgewählt wird. Belässt man alle Parameter unverändert, befindet sich der Roboter in der Mitte des Szenarios und kann, wenn er mit einem Controller ausgestattet wird, sofort eingesetzt werden. Erweitert man das Szenario um einige Hindernisse, die alle als entsprechend definierte `Solid`-Knoten dargestellt werden, so erhält man ein fertiges Testumfeld für den Khepera-II-Roboter.

Die visuelle Darstellung erfolgt mittels OpenGL in ansehnlicher 3D Graphik und ist somit weitestgehend Plattformunabhängig. Da die Graphik die Geschwindigkeit der Simulation begrenzt kann sie zur Maximierung der Ausführungsgeschwindigkeit ausgeschaltet werden.

In Webots gibt es passive und aktive Objekte, die sich dadurch unterscheiden, dass die Aktiven im Gegensatz zu den Passiven ein Steuerprogramm ha-

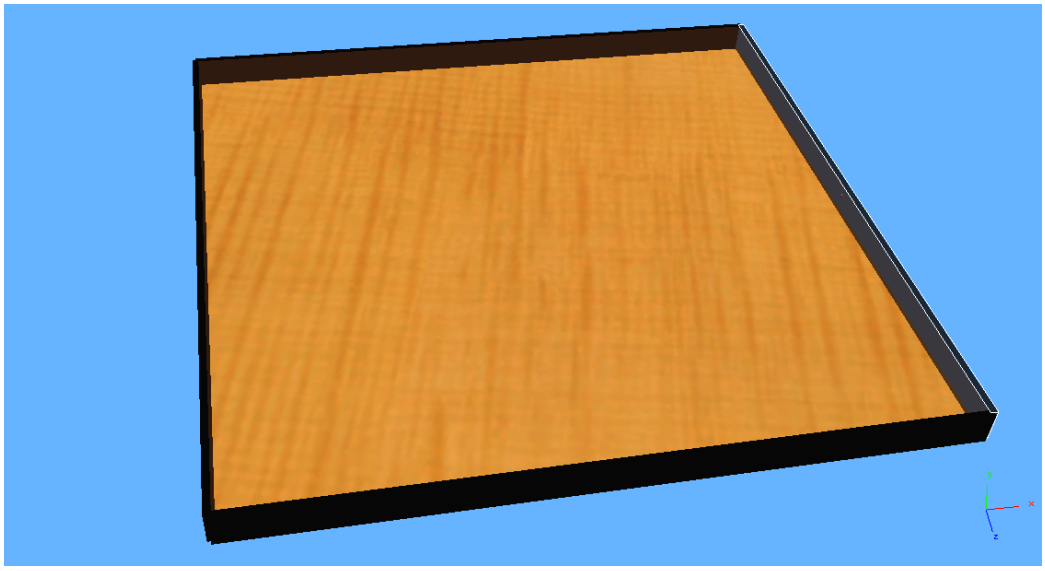

Abbildung 2.5: Webots: Bodenplatte und Wände

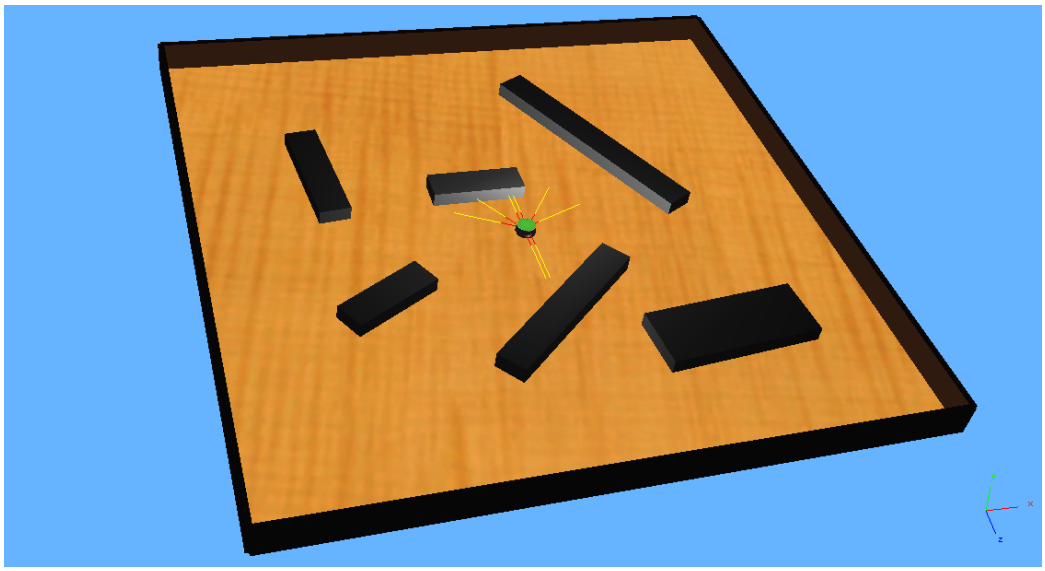

Abbildung 2.6: Webots: Khepera-II in Umgebung

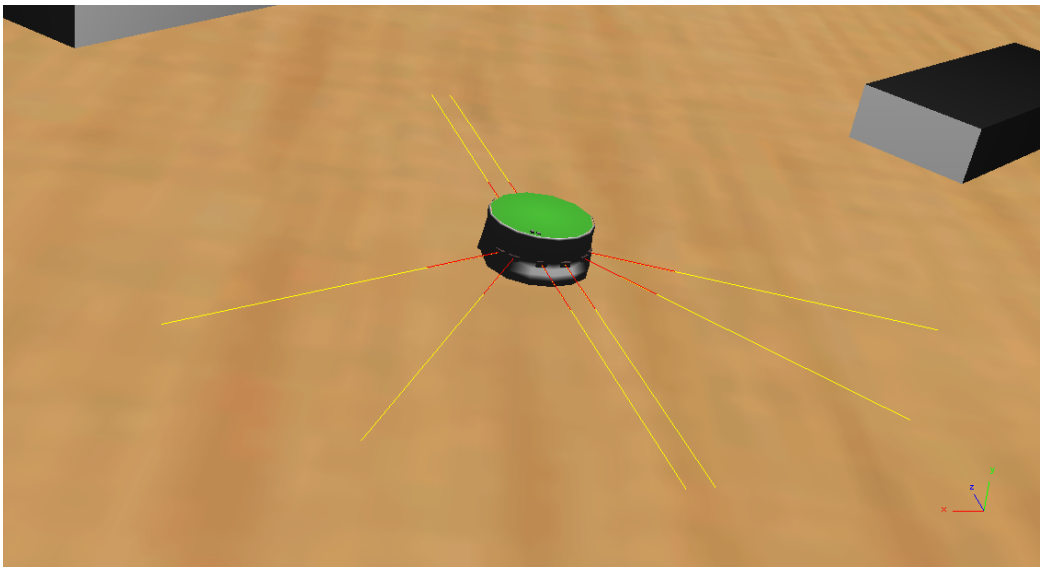

Abbildung 2.7: Webots: Khpera-II Roboter

ben. Es muss nicht jedes Objekt, das mit einem Controller ausgestattet ist, eine Art Roboter sein. Das Objekt könnte auch andere Aufgaben übernehmen; im Rahmen dieser Arbeit werden aktive Objekte als Funkleuchttürme verwendet. Ihre einzige Aufgabe ist, in regelmässigen Abständen einen bestimmten Identifizierungscode zu senden.

Als Besonderheit ist der Supervisor anzusehen, da er zwar einen Controller, nicht aber eine simulierte physische Präsenz haben muss. Er ist die einzige Instanz, die beliebige Änderungen an anderen Objekten und an den Eigenschaften der Welt vornehmen kann. Er kann beispielsweise Screenshots speichern, Texte darstellen, Controller starten und stoppen oder Objekte manipulieren, indem er sie verschiebt oder ihre Farbe ändert. Es ist grundsätzlich möglich, mehreren Objekten einen Controller zuzuweisen und Kommunikation über verschiedene Medien zu simulieren. Somit lassen sich Multi-Roboter-Szenarien simulieren.

Die Programmierung der Robotercontroller kann in C, C++ oder Java erfolgen oder mit einem beliebigen anderen Werkzeug, wenn Ein- und Ausgabeinformationen über TCP/IP kommuniziert, der simulierte Roboter also ferngesteuert wird. Webots lädt die kompilierten Steuerprogramme

beim Start oder Neustart der Simulation. Diese laufen immer nach dem gleichen Schema ab: zunächst wird eine Erststart-Routine ausgeführt, dann eine Reset-Funktion und schliesslich bis zum Ende der Simulation die Run Methode. Die Run Methode bekommt die Laufzeit des Roboters zur Aufrufzeit als Parameter übergeben, damit zeitabhängige Operationen auch bei unregelmässigem Aufruf des Controllers möglich sind. Die Simulation kann jeweils nur dann fortgesetzt werden, wenn alle Controller ihren Durchlauf beendet haben. Benötigt der Controller mehr Zeit für seine Bearbeitung als in einem „normalen“ Simulationsschritt verfügbar ist, verlangsamt sich die Simulation oder steht zeitweise still.

2.4 Grund-Design

2.4.1 Aufgabenanalyse

Im folgenden Teil wird erläutert, wie die Aufgabenstellung „Entwicklung einer planenden Robotersteuerung“ angegangen wird und das gestellte Problem mit den gegebenen Mitteln und Methoden bearbeitet werden kann. Die Aufgabenstellung, die dem zuvor beschriebenen „rescue szenario“ entstammt, stellt sich wie folgt dar:

- Multiple Agenten mit spezialisierten Expertisen
- Umgebung mit einer weitestgehenden Bewegungsfreiheit für die Agenten
- Landmarken sind bereits Teil der Umgebung
- Landmarken sind immer eindeutig identifizierbar auch durch verschiedene Agenten
- Die Agenten sind Roboter vom Typ Khepera II; um die Entwicklung zu erleichtern wird Webots zur Simulation verwendet
- Die Planung und Informationsspeicherung erfolgt symbolisch

- Agenten planen für sich selbst und koordinieren sich soweit wie möglich
- Zum Vergleich der Effizienz kann auch ein zentraler Plan erzeugt werden

Die gegebene Aufgabenstellung, lässt sich zweckmässig auf die folgenden Teilaufgaben aufteilen:

1. Erzeugung einer Umgebung in Webots
2. Umsetzung eindeutig identifizierbarer Landmarken
3. Bewegungs-Steuerung der Agenten
4. Zielgerichtete Planung von Aktionen
5. Kommunikation zwischen den Agenten
6. Koordination von Plänen
7. Umsetzung einer zentralen Planungsstelle

Zu diesen Teilaufgaben kommt jeweils eine Visualisierung der für den Benutzer relevanten Informationen hinzu.

Im Folgenden werde ich die einzelnen Teilaufgaben genauer analysieren und mögliche Lösungsansätze vorstellen.

Erzeugung einer Umgebung in Webots Webots eignet sich hervorragend zur Erzeugung von Umgebungen, insbesondere wenn diese in abstrakter Form aus Kombinationen von einfachen 3-dimensionalen geometrischen Körpern gestaltet werden.

Um die Umgebung leicht variierbar zu halten, wird sie aus einer Anzahl von Quadern konstruiert. Im Hinblick auf bessere Anschaulichkeit wäre es natürlich „schöner“, den Objekten mehr Gestalt zu geben, z.B. Hindernisse als Häuser oder Autos darzustellen. Hält man sich dabei an die Quaderform der Hindernisse kann man weiterhin einen Quader als *Bounding_Object* verwenden, wodurch sich nichts an der Simulation selbst verändert. Da hierdurch

aber die einfache Veränderbarkeit der Umgebung eingeschränkt wird, wurde davon Abstand genommen.

Die Grundfläche der Landschaft ist, im Hinblick auf die spätere Umsetzung auf einen Khepera-II Roboter auf 2*2-Quadratmeter beschränkt. Da die Sensoren des Khepera-II nur Hindernisse erkennen können, gibt es keine Vertiefungen, z.B. Gräben oder Bordsteinkanten und nur eine Ebene, in der sich das ganze Szenario abspielt. Die Webots-Welt ist des weiteren statisch - sie verändert sich nicht von alleine - und, zumindest in der Simulation, deterministisch - die Auswirkungen aller Aktionen sind eindeutig vorhersehbar. Da die Bewegung des Roboters von einer Physik-Engine berechnet wird, ist sie nicht völlig frei von Störungen und nicht jeder Steuerbefehl wird exakt umgesetzt. Die Bewegungsfreiheit der Roboter auf der Karte und die Tatsache, dass ein Khepera-II auf der Stelle drehen kann, führen ausserdem dazu, dass jede Bewegungsaktion revidierbar ist. Dies führt dazu, dass es keine Sackgassen gibt und jeder einmal erreichte Ort garantiert wieder erreichbar ist, auch für andere Agenten. Ausserdem wird die Landschaft so gestaltet, dass jeder Punkt auf der Karte für alle Agenten gleichermassen zugänglich ist; es gibt also keine Abtrennung einzelner Bereiche. Durch die geringe Sensorreichweite der Roboter und die zweifelhafte Präzision ihrer Bewegungen ist es des weiteren erforderlich, dass Landmarken recht nahe beieinander positioniert werden und, so eine Verbindung zwischen je zwei Landmarken erwünscht ist, sich zwischen ihnen keine Hindernisse befinden.

Die Simulation des Rescue-Szenarios in Webots ist Teil der Aufgabenstellung und wird daher nicht in Frage gestellt. Es sei jedoch angemerkt, dass von den umfangreichen Fähigkeiten der Simulation nur ein geringer Anteil benötigt wird. So bewegen sich die Agenten beispielsweise nur in zwei Dimensionen. Auch werden viele Merkmale der Physik-Engine nicht genutzt. Zur Beschleunigung der Simulation lässt sich die graphische Darstellung ausblenden; somit steht dann nahezu die gesamte Rechenleistung für die Steuerprogramme zur Verfügung.

Umsetzung eindeutig identifizierbarer Landmarken Landmarken sollen für die Khepera-II-Roboter eindeutig identifizierbar sein. Leider ist dieser

Roboter standardmässig nur mit Lichtsensoren ausgestattet, die in erster Linie der Hinderniserkennung und Lichtquellenverfolgung dienen. Es ist sehr aufwändig, aus einem Abstandssensorbild eine eindeutige Landmarke zu erzeugen und nahezu nicht möglich mit Sensoren, die eine so geringe Reichweite und Präzision haben wie die des Khepera-II. Landmarken auf einem Kamerabild erkennen zu können wäre ideal, da sich so hohe Sichtweiten erreichen liessen. Bildverarbeitung ist allerdings sehr rechenintensiv und das eindeutige Identifizieren eines Objektes ist selbst noch Gegenstand aktueller Forschung und kann daher nicht genutzt werden. Möchte man trotzdem Kameras einsetzen, würde sich eine Farbkodierung der Landmarken anbieten.

Um dennoch ein verlässliches und gleichzeitig praktisch umsetzbares Identifikationsmittel zu nutzen, wurde entschieden, die Landmarken als Leuchtfeuer umzusetzen. Jede Landmarke sendet ihren Identifikationscode mit einer eingeschränkten Reichweite, so dass jeder Agent, der in den Sendebereich gelangt, diese Information empfängt. Der Vorteil dieser Variante ist, dass hierfür bereits verfügbare Hardware - der *Radioturret* des Khepera-II - genutzt werden kann. Leider ist keine Peilung möglich, die Landmarke ist aus Sicht des Roboters also die gesamte Sendefläche. Für eine reine Landmarkenbasierte Navigation wäre dies möglicherweise schon ausreichend, da bei hinreichend präziser Navigation die jeweils nächste Landmarke angesteuert werden kann. Um garantiert brauchbare Navigationsergebnisse in der freien Umgebung erzielen zu können muss der Agent in der Lage sein, das Sendezentrum, also die Landmarke selbst, orten zu können. Nur so sind aussreichend präzise Karten anfertigbar und ist die Position des Agenten auf dieser Karte bestimmbar. Eine auch in einem nicht simulierten Szenario anwendbare Methode wäre beispielsweise, im Zentrum der Landmarke eine zweite Markierung aufzustellen, die durch die Abstandssensoren gefunden werden kann. Positioniert man etwa ein zylindrisches Objekt - in der Praxis der Sender des Leuchtfeuers, in der Simulation ein möglichst kleines, von den Sensoren jedoch gut erkennbares Hindernis -, so kann der Agent dieses systematisch in der Landmarkenfläche suchen und somit ein recht genaues Zentrum bestimmen. Diese Methode ermöglicht jedoch nur einen Abgleich der x- und y-Koordinaten des Roboters, nicht jedoch eine Aussage über dessen Ver-

drehung. Ausserdem sind der Szenariogestaltung dadurch gewisse Grenzen gesetzt, da sich a) kein zweites Hindernis im Bereich der Landmarkenfläche befinden darf und b) im Zentrum dieser Fläche immer ein Hindernis steht, was beispielsweise die Markierung schmaler Durchgänge als Landmarken erheblich erschwert. Aus diesen Gründen - und aus Gründen der erheblich einfacheren Umsetzung - wurde beschlossen, die Information über die Position des Zentrums einer Landmarke dem Roboter als zusätzliches Wissen mitzuteilen. Befindet sich der Roboter im Sendebereich einer Landmarke kann er nun die Position ihres Zentrums abrufen, zu seiner derzeitigen eigenen Position in Relation setzen und somit zu seinen Zwecken nutzen. Obgleich dieses Verfahren die Umsetzung auf einen echten Roboter erschwert, ist es im Grunde nicht unrealistisch, da normalerweise das Anpeilen eines Leuchtfeuers - und damit die Bestimmung seiner Position - technisch unaufwändig ist.

Steuerung der Agenten Die Steuerung der Bewegung des Roboters erfolgt je nach dessen derzeitiger Einsatzphase entweder zielgerichtet als Teil der Abarbeitung eines Plans oder im Rahmen der Erkundung. Ausserdem ist die Kollisionsvermeidung permanenter zentraler Bestandteil der Bewegungs-Steuerung des Roboters.

Abhängig davon, wie gut, vollständig und analysierbar die Karte des Roboters ist, auf der er seine eigenen Bewegungen nachvollzieht, lässt sich die Kollisionsvermeidung vorausschauend oder situativ reaktiv umsetzen. Vorausschauende Kollisionsvermeidung erfordert, dass Hindernisse, noch bevor sie durch die Sensorik als solche erkannt werden, in die Entscheidungen der Steuerung einbezogen werden. Es wird eine Route berechnet, die an Hindernissen vorbei oder um sie herum führt. Neben der traditionellen Routenplanung gibt es andere Verfahren, die sich dieses Problems annehmen. Reaktive Kollisionsvermeidung hingegen basiert ausschliesslich auf den aktuellen Sensorinformationen. Melden die Sensoren beispielsweise ein Hindernis direkt vor dem Roboter, kann dieser zu einer Seite abdrehen. Anhand einiger Regeln kann so, bei ausreichender Sensorabdeckung der näheren Umgebung, eine Kollision mit einem statischen Hindernis immer verhindert werden. Dabei ist

noch zu beachten, dass die Khepera-II-Roboter sich verhältnismässig langsam bewegen, nahezu keinen Bremsweg haben und in der Lage sind, auf der Stelle zu drehen, was die reaktive Kollisionsvermeidung erheblich vereinfacht.

Zielgerichtete Navigation lässt sich, ebenfalls von der Detailtreue der Karte abhängig, entweder geplant oder trivial durchführen. Bei geplanter Navigation berechnet der Roboter einen Weg an den ihm bekannten Hindernissen vorbei. Bei trivialer, ungeplanter Navigation steuert der Roboter, ausgestattet mit Positions- und Richtungsangaben auf das Ziel zu. Hindernisse werden dann mit bestimmten Strategien oder schlicht durch Kollisionsvermeidung umgangen. Dies führt nicht zwingend zum Ziel erfordert aber einen geringeren Rechenaufwand und eignet sich für die Erkundung unbekannter Umgebungen. In der hier beschriebenen Arbeit wird triviale Navigation zwischen zwei benachbarten Landmarken eingesetzt.

Die Erkundung der Umgebung kann randomisiert oder systematisch erfolgen. Auch diese Entscheidung ist wiederum von der Qualität der verfügbaren Karte abhängig. Randomisierte Verfahren eignen sich möglicherweise besser für freie Umgebungen mit vielen Zielen der Suche, während systematische Verfahren sich eher für die sorgfältige Suche nach einem bestimmten Ziel eignen. Mittels systematischer Verfahren ist auch eine vollständige Erkundung der gesamten Umgebung möglich, somit das gesicherte Auffinden aller vorhandenen Ziele, was durch randomisierte Verfahren nicht gewährleistet werden kann. Bei der systematischen Erkundung kann noch zwischen gezielter und ungezielter systematischer Erkundung unterschieden werden. Bei gezielter systematischer Erkundung wird anhand der Karte entschieden, welche Bereiche zu erkunden sind, bei ungezielter systematischer Erkundung braucht die Karte dagegen nicht beachtet zu werden. Ohne Karte und ohne Feststellung der eigenen Position kann jedoch auch eine systematische Suche nicht garantieren, dass alle Ziele gefunden werden.

Planung von Aktionen Der eigentliche Kern dieser Arbeit ist das Planen und Koordinieren der Handlungsabfolgen mehrerer Agenten bei der Bearbeitung eines gemeinsamen Problems: das Löschen aller Brände und die Rettung aller Verletzten. Vor der Planung steht die Definition aller Aktionen, die Teil

des Plans sein sollen. Offensichtlich sind zumindest die Aktionen „löschen“ und „retten“ erforderlich und offensichtlich sind diese Aktionen an die Bedingung 1. der Übereinstimmung der Positionen des Agenten und des Problems sowie 2. der Fähigkeit des Agenten, diese Aktion auszuführen, gebunden. In der verwendeten vereinfachten STRIPS-Sprache gibt es keine „NOT“ Information für Prädikate und es gilt die „closed world assumption“; das heisst, ist ein Prädikat in einem Zustand vorhanden gilt es; alle anderen Prädikate gelten nicht.

Zum Zwecke der Navigation gibt es die „Move“-Aktion die den Agenten von einer Position zu einer anderen bewegen lässt. An dieser Stelle tut sich die Frage auf, ob es hinsichtlich der Effizienz des ganzen Verfahrens zweckmässig ist, die Navigation von der Planung der übrigen Handlungen zu trennen. Der Vorteil einer solchen Trennung wäre sicherlich, dass sich für die Navigation erheblich effizientere Planungsverfahren - wie beispielsweise der A*-Algorithmus - verwenden liessen. Der Nachteil hingegen ist, dass, wenn - eigentlich vorhandene - Informationen nicht in die Planung einfliessen - hier der Pfad zwischen zwei Problemen - sich die Optimalität des Planes nicht mehr gewährleisten lässt. Es wären auch bereits während der Planung weitere Untersuchungen der Verbindungen zwischen Landmarken - beispielsweise Heuristiken oder tatsächliche Pfade - notwendig, um wenigstens Vollständigkeit und Korrektheit eines entsprechenden Planungsverfahrens zu erreichen.

Sind die Aktionen erst beschrieben und ein System zum Wissenserwerb vorhanden, lässt sich mit einem der zuvor beschriebenen Planungsverfahren ein Plan erzeugen. Alternativ zu diesen Planungsverfahren zur Erzeugung vollständiger Pläne über alle Ziele wäre auch ein Ansatz denkbar, der die Ziele getrennt von einander betrachtet. Welches Ziel ausgewählt wird, ist von einer Bewertung aller Ziele abhängig. In diese Bewertung fliessen beispielsweise karthesischer Abstand, Nähe zu anderen Zielen und vermutete Bewertung durch andere Agenten ein. Auf diese Weise liessen sich die Pläne auf die Erfüllung einzelner Ziele beschränken, was den Erzeugungsaufwand erheblich reduziert. Dieser Ansatz ist nur möglich, wenn das Erfüllen eines Ziels die Erfüllbarkeit der anderen Ziele nicht einschränkt oder diese Abhängigkeit

bekannt oder feststellbar ist.

Im Abschnitt 2.6 werde ich noch ausführlicher auf die verwendeten Ansätze und Verfahren dieser wichtigen Grundlage dieser Arbeit eingehen.

Kommunikation zwischen den Agenten Die Kommunikation zwischen den Agenten dient dem Informationsaustausch und der Koordination. Zur technischen Realisierung der Kommunikation wird der „Radio Turret“ des Khepera-II genutzt.

Neben der Landmarken basierten Karte, auf die sich die Planung stützt, erzeugt jeder Agent eine Hinderniskarte seiner Umgebung. (s. 2.5.2) Es ergibt sich dabei ein neues Problem, welches ebenfalls mit der eingeschränkten Sensorfähigkeit des Khepera-II zusammenhängt. Es ist für den Roboter nicht möglich, ein Hindernis von einem anderen Roboter zu unterscheiden. Da demzufolge ein Roboter als Hindernis identifiziert wird, werden die Karten verfälscht sobald sich zwei Agenten begegnen und somit unbrauchbar. Daher wird eine ebenfalls in der Reichweite eingeschränkte Funkanlage eingesetzt, um anderen Robotern mitzuteilen, dass sie sich gerade in Sensorreichweite eines Roboters befinden und sie deshalb ihre Karte nicht länger genau erstellen können. Das Erzeugen der Karten wird ausgesetzt, solange die Agenten sich gegenseitig stören. Sobald aussreichend globale Informationen verfügbar sind wäre es auch möglich, die Positionen der Roboter gegenseitig auszutauschen und auf diese Weise festzustellen, ob das erkannte Hindernis ein anderer Roboter ist oder nicht. Solange aber noch keine gemeinsamen Informationen vorliegen ist die gegenseitige explizite Identifizierung der Agenten wegen der eingeschränkten Sensorfähigkeiten der Roboter die einzige Möglichkeit, dieses Problem zu umgehen.

Der Hauptzweck der Kommunikation zwischen den Agenten ist - wie bereits erwähnt Informationsaustausch und Koordination. Die „Radio Turrets“ sind in der Simulation mit endloser Reichweite ausgestattet und ausserdem störungs- und fehlerfrei, was für Funkkommunikation eine Idealisierung darstellt. Auch dass Hindernisse keinen Einfluss auf die Funkverbindung haben, ist eine Vereinfachung des Szenarios.

Da Webots für die Simulation die Zeit in kurze Abschnitte, sogenann-

te „Ticks“ einteilt, erfolgt die Kommunikation schrittweise. Das heisst, die Nachricht wird in einem bestimmten Tick vom Sendepuffer des einen Roboters in den Empfangspuffer des/der Anderen verschoben. Wie häufig und wie schnell dies geschieht, hängt von den Einstellungen der Funkkomponente ab. Ein nicht unerhebliches Problem ist die Anzahl der beteiligten Agenten. Da sie alle auf dem selben Kanal senden, würden sie sich eigentlich gegenseitig stören. Auch die Menge der gesendeten Informationen steigt natürlich mit der Anzahl der Agenten. Um dennoch Fehlerfreiheit zu ermöglichen, wurde der Empfangspuffer entsprechend gross gewählt, so dass sichergestellt ist, dass keine Nachrichten verloren gehen.

Für ein flexibles Multi-Agenten-System ist es erforderlich, dass die Kommunikation die folgenden Anforderungen erfüllt:

- Fehlerfreiheit - Gesendete Nachricht entspricht der Empfangenen
- FIFO-Reihenfolge - Nachrichten werden in der Reihenfolge empfangen in der sie gesendet wurden
- Broadcastfähigkeit - Nachrichten an alle Agenten erleichtert Informationsverbreitung
- Peer-To-Peer Fähigkeit - Nachrichten mit definiertem Sender und Empfänger ermöglichen gezielte Kommunikation

Das verwendete Kommunikationsmedium „Funk“ erfüllt diese Bedingungen, mit Ausnahme der Broadcastfähigkeit von sich aus nicht. Es bedarf eines geeigneten Kommunikations-Protokolls, um alle geforderten Eigenschaften erfüllen zu können. Es gibt hinreichende Standardverfahren, um diese Eigenschaften auf unsicheren Broadcastmedien zu ermöglichen. Da die simulierte Kommunikation bereits die Fehlerfreiheit und FIFO-Reihenfolge mit sich bringt, ist nur noch die eindeutige Adressierung der einzelnen Agenten zu klären.

Da die Agenten in einem möglichst allgemeingültigen Szenario nur geringe Informationen über andere Agenten haben, kann nicht angenommen werden, dass jeder Agent von Anfang an jeden anderen kennt. Die Agenten besitzen

auch keine Informationen über Anzahl oder Art (Fähigkeiten..) der anderen beteiligten Agenten. Man stelle sich beispielsweise eine Ansammlung unterschiedlich spezialisierter Agenten vor, die sich zur Lösung eines Problems zusammenfinden müssen.

Zur eindeutigen Adressierung wird daher unterstellt, dass jeder Agent einen eindeutigen Namen hat, aus dem sich eine entsprechende Adresse erzeugen lässt. Auch ohne solche Massnahmen liessen sich Adressen an die Agenten vergeben. Für dynamische Gruppenorganisation gibt es hinreichende Verfahren, auf die hier nicht weiter eingegangen werden soll.

Mögliche Inhalte der Kommunikation im gegebenen Szenario sind:

- Neue Landmarke entdeckt; Identifikationsinformationen dieser Landmarke
- Pfad zwischen zwei Landmarken gefunden bzw. bestätigt
- Anfrage nach Positionen von Landmarken relativ zu zwei gemeinsam bekannten Landmarken
- Informationen über die Position von Landmarken relativ zu zwei Örientierungslandmarken"
- Koordinationsinformationen
- Speziell für die zentrale Planung: Position des Agenten (Landmarke)

Koordination von Plänen Mit den gesammelten und kommunizierten Informationen über die Umgebung erzeugt jeder Agent für sich einen (Teil-)Plan, der der Lösung des Gesamtproblems dient. Bereits vor der Erstellung dieses Plans jedoch stehen Aspekte der Koordination. So überprüfen die Agenten beispielsweise zunächst, welche Probleme sie selbst bearbeiten können und teilen dies den anderen Agenten mit. Ist ein Problem nur von mehreren Agenten lösbar, schätzen diese ihren jeweiligen Aufwand ab und verteilen die Teilprobleme dementsprechend. Anhand dieser Teilprobleme werden dann die jeweiligen Pläne erzeugt. Hat jeder Agent für sich, anhand der ihm zur Verfügung stehenden Informationen einen Plan erzeugt,

müssen diese Pläne koordiniert werden, um die Korrektheit des Gesamtplans sicher zu stellen. Auch das Formulieren der Einzelprobleme als Aufgaben, die durch einen Agenten bearbeitet werden können, zählt zur Koordination. Wichtig ist in diesem Zusammenhang die Betrachtung des Verhältnisses von Aufwand und Nutzen, insbesondere, in wieweit sich der Plan durch wieviel Kommunikation verbessern lässt. In dem gestellten Szenario gibt es keine exklusiven Ressourcen, da sich aufgrund der freien Navigation Platzkonflikte durch spontane, reaktive Ausweichmaneuver leicht beheben lassen. Die Hauptaufgabe der Kommunikation wird also sein, die Ziele möglichst effizient auf die Agenten zu verteilen und so den Gesamtplan zu verbessern. Im Abschnitt „Koordination“ wird noch aussführlicher auf die Umsetzung des Koordinationsverfahrens eingegangen.

Umsetzung einer zentralen Planungsstelle Vergleicht man verteilte Planung mit zentraler Planung, findet man leicht mehrere Vorteile auf Seiten der verteilten Planung, die diesen Ansatz sehr interessant machen. So wird die verfügbare Rechenkapazität mehrerer Agenten genutzt, was die Planung beschleunigen sollte. Auch die Reduzierung des Wissens, und die damit einhergehende Einschränkung des Suchraums jedes einzelnen Agenten beschleunigt die Plansuche. Die hohe Spezialisierung einzelner Agenten erleichtert des weiteren das Finden und Nutzen von Heuristiken und Strategien für bestimmte Aufgaben. Und schliesslich ermöglicht die verteilte Planung, Wissen zu dezentralisieren, Interessen der einzelnen Agenten in den Plan einzubeziehen, von denen andere nichts erfahren müssen oder sollen, sowie die Ausfallsicherheit durch redundante Einheiten zu erhöhen.

Obwohl es also eine Reihe guter Gründe für verteilte Planung gibt, bringt diese jedoch auch einige zum Teil nicht unerhebliche Schwierigkeiten mit sich: Zum einen gibt es kein Verfahren, welches für jedes beliebige Planungsproblem vollständig ist. Insbesondere bei Problemen, die „echte“ Kooperation erfordern, tauchen hier Schwierigkeiten auf. Zum anderen existiert kein Verfahren, das garantiert einen optimalen Plan findet. Da zu keinem Zeitpunkt ein Algorithmus über die vollständigen Informationen verfügt, kann es auch kein optimales verteiltes Planungsverfahren geben. Die Vor- und Nachteile

zentraler und verteilter Planung sind daher für jeden Einzelfall erneut abzuwägen. Im Rahmen dieser Arbeit wird daher zusätzlich zu der verteilten Planung ein zentraler Plan erzeugt und mit dem verteilt erzeugten verglichen. Die zentrale Planungsstelle benötigt hierzu vollständige Informationen über alle Agenten, also Beschreibungen der Aktionen, Wissen jedes Agenten und alle Ziele. Genaueres zur Umsetzung der zentralen Planung wird im Abschnitt 2.8 beschrieben.

2.4.2 Implementierungsansatz

Die Steuerung der - eigentlich die Kontrolle über die - Agenten erfolgt, dem Agentenprinzip entsprechend durch jeden Agenten selbst. Sie folgt grundsätzlich einem Multi-Controller Konzept, in dem die Datenhaltung und die Schnittstelle zum Roboter von den Klassen *Khepera* und *Robot* übernommen werden. Die eigentliche Steuerung wird von speziallisierten *Controller* Klassen durchgeführt.

Die Klasse *Khepera* ist die zentrale Steuerungsklasse. Sie ruft in jedem Durchlauf der Simulation alle Steuerklassen und die Aktualisierungsfunktionen des Roboters auf. Die Klasse *Robot* ist die Sammlung der Fähigkeiten und Zustandsinformationen des Agenten. Sie verwaltet die Sensoren sowie die Aktoren des Agenten. Alle Steuerklassen haben freien Zugriff auf dieses Objekt; sie können Zustandsvariablen verändern oder Steuerbefehle absetzen.

Alle Steuerklassen erben von der Klasse *Controller* und werden dem *Khepera* Objekt mittels der `set_controller` Funktion zur Verwaltung übergeben, so dass je nach Bedarf eine beliebige Kombination von *Controller* für einen bestimmten Agententyp genutzt werden kann. So lsst sich beispielsweise in einer Konfigurationsdatei zentrale oder verteilte Planung einstellen. Je nach Einstellung werden dann den Agenten und dem Supervisor andere *Controller* zugewiesen. Im Grunde kann eine beliebige Anzahl verschiedener *Controller* zur Steuerung eingesetzt werden. Diese werden der Reihe nach abgearbeitet, haben freien Zugriff auf gemeinsame Steuerinformationen und können nach Belieben Steuerbefehle erteilen. Die Steuerbefehle beschränken sich auf die

Konkreten, also auf die Bewegung des Roboters und z.B. auf die Ausgabe von Textnachrichten. Die abstrakten Aktionen, die später noch erläutert werden, sind Bestandteil eines Controllers und somit nicht global verfügbar. Die Bewegungssteuerung erfolgt durch Setzen von Variablen der Drehgeschwindigkeit des linken bzw. rechten Rades. Diese Informationen werden am Ende eines Steuerungslaufes, nachdem alle Controller bearbeitet wurden, zum Setzen der entsprechenden Parameter in der tatsächlichen Robotersteuerung genutzt. Es ist ersichtlich, dass, wenn mehrere Controller versuchen, eine ihrer Auffassung nach „richtige“ Geschwindigkeit einzustellen, widersprüchliche Steuerbefehle entstehen können. Um dennoch die Nutzung mehrerer Controller zu ermöglichen, wurde die Kollisionsvermeidung aus den restlichen Controllern ausgegliedert und hat immer Vorrang. Auf diese Weise ist sichergestellt, dass kein anderer Controller den Roboter „gegen die Wand“ steuert. In der Nähe von Hindernissen wird dadurch allerdings die zielgerichtete Navigation etwas eingeschränkt. Im Allgemeinen muss bei der Entwicklung eines Controllers darauf geachtet werden, dass keine widersprüchlichen Befehle gegeben werden.

Es sei an dieser Stelle angemerkt, dass sich eine solche Multi-Controller-Steuerung ebenfalls als Multi-Agenten-System auffassen lässt und Konflikte und widersprüchliche Absichten bei der Steuerung eines Roboters ideale Kandidaten für Koordination sind. Da in diesem Szenario die Aufgaben des Agenten jedoch nur entweder Erkundung oder Abarbeitung eines Plans sind und weitere Controller in erster Linie der Kommunikation dienen, werden durch das Herauslösen der Kollisionsvermeidung aus der Robotersteuerung eventuell mögliche Konflikte von vorn herein vermieden.

2.4.3 Wissensrepräsentation

Ein wichtiger Bestandteil der Planung ist die Wissensrepräsentation. Diese erfolgt symbolisch in einer STRIPS-artigen Sprache.

Fakten bestehen aus einem eindeutigen Identifizierungscode und einer festen Anzahl von Parametern. In der Beschreibung des jeweiligen Faktes ist der Identifizierungscode eine Zeichenkette. Diese wird zur Laufzeit aus Speicher

und Rechenzeitgründen einem `Long Integer` durch *hashen* zugeordnet. Die Parameter eines Faktes sind in seiner Beschreibung entweder `String` oder `Long Integer` und werden intern immer als `Long Integer` gespeichert. War der Parameter ursprünglich ein `String` wird diese Information gespeichert. Zum Zwecke der Sequentialisierung oder der Ausgabe eines Faktes werden den gehashten Werten wieder Zeichenketten zugeordnet. Durch die Verwendung von `Long Integer` als Typ der Identifizierung und der Parameter werden Operationen zum Vergleichen zweier Fakten einfacher und auch grosse Mengen von Fakten schnell sortier- und durchsuchbar. Ausgeschrieben besteht ein Fakt aus einem Bezeichner, der Anzahl seiner Parameter und der Liste seiner Parameter.

Das Wissen, das der Agent „khepera2_0“ über seine eigene Position auf der topologischen Karte hat, sieht ausgeschrieben beispielsweise so aus:

```
at 2 khepera2\_0 landmark\_07
```

Bereits ehe die Agenten beginnen, Wissen zur Laufzeit zu erwerben, steht ihnen einiges an initialem Wissen zur Verfügung. In der Konfigurationsdatei des jeweiligen Roboters, in der dieses Wissen enthalten ist, ist es mit —knowledge— markiert. Der Roboter weiss somit, dass er das nachstehende Faktum in den Wissensspeicher - den aktuellen oder initialen Zustand - aufnehmen soll.

Der Agent mit der Bezeichnung „khepera2_0“ besitzt die Fähigkeit der Feuerbekämpfung. In seiner Konfigurationsdatei befindet sich somit die folgende Zeile: `knowledge firefighter 1 khepera2_0`

Mehrere Fakten bilden zusammen einen Zustand. Jeder Fakt kann nur genau einmal in einem Zustand vorhanden sein. Fakten können einem Zustand hinzugefügt oder aus diesem entfernt werden. Es sind Operationen zum Vergleich zweier Zustände definiert. Neben „Gleichheit“ ist vor allem „part_of“ wichtig. Dieser Operator überprüft, ob ein Zustand Teil eines Anderen ist.

Aktionen sind zunächst als Operatoren definiert und enthalten Variablen. Durch *matchen* auf einen gegebenen Zustand lassen sich für einen Operator Belegungen der Variablen finden, so dass gültige Aktionen entstehen. Wie in STRIPS üblich, ist ein Operator über seinen Bezeichner und eine feste

Anzahl Parameter definiert. Die Anforderungen des Operators sind in Form von Fakten, die wiederum Variablen enthalten, beschrieben. Entsprechendes gilt für die Auswirkungen des Operators auf den aktuellen Zustand.

Die Operatoren werden aus Dateien ausgelesen und geparst. Um dem Parser die Arbeit zu erleichtern wird folgende einfache Sprache verwendet:

Operatoren beginnen mit dem Bezeichner gefolgt von der Arität und einer entsprechenden Anzahl Variablennamen. Variablennamen beginnen immer mit einem „`:`“ um sie von anderen Zeichenketten zu unterscheiden. Ihre Anzahl entspricht der Arität und es müssen alle im Operatorrumpf verwendeten Variablen im Operatorkopf definiert sein. Als nächstes sind die Anforderungen beschrieben. Dies wird durch `:pre:` (*precondition*) markiert. Die Vorbedingungen selbst sind wie Fakten definiert, ausser dass sie Variablen enthalten können, die im Operatorkopf definiert sind. Nach den Vorbedingungen folgen entsprechend die Add- und die Del-Effekte, markiert durch `:add:` bzw. `:del:`. Das Ende eines Operators ist durch `:end:` markiert.

Die Beschreibung des Move-Operators sieht in dieser Sprache dann so aus:

```
move 3 :who :from :to
  :pre:
    at 2 :who :from
    landmark_pos 1 :to
    path 2 :from :to
  :add:
    at 2 :who :to
  :del:
    at 2 :who :from
:end:
```

Der Agent erwirbt während seiner Erkundungen oder durch Kommunikation neues Wissen. Bei der Erkundung werden die erworbenen Informationen in ein entsprechendes Format umgewandelt und in den Wissensspeicher aufgenommen. Die von anderen Agenten empfangenen Fakten sind entsprechend als `String` serialisiert und können geparst und anschliessend in den

aktuellen Zustand aufgenommen werden. Da möglichst alle Informationen als sequentialisierte Zeichenkette übertragen und ausgegeben werden sind Faktenübertragung, Zustände und Aktionen sowie erzeugte Pläne leicht nachvollziehbar.

2.5 Karten

2.5.1 Topologische Karte

In dem gegebenen Szenario bewegen sich die Agenten so frei wie möglich in ihrer Umgebung. Jede Bewegung soll dazu dienen, das Gesamtsystem der Lösung des Problems näher zu bringen. Zu Beginn der Bearbeitung wissen die Agenten nichts von ihrer Umgebung. Sie müssen sich diese Informationen erst noch aneignen. Hierzu dient zum Einen die Erkundung der Umgebung, zum Anderen der Informationsaustausch zwischen den Agenten, z.B. über entdeckte Landmarken.

Es sind feste Landmarken vorgegeben, die von allen Agenten immer eindeutig zu identifizieren sind. Sie sind die Grundlage der Navigation der Agenten und stellen auch die einzige Information dar, die die Roboter untereinander über ihre Umgebung austauschen müssen. Mehrere Landmarkenkarten sind verhältnismässig einfach kombinierbar. Es werden lediglich zwei bekannte Landmarken benötigt, die in beiden Karten enthalten sein müssen. Relativ zu diesen Landmarken können die Positionen aller anderen auf beiden Karten vorhandenen Landmarken berechnet werden. Landmarken dienen auch der Positionsbestimmung der Roboter. Da sie immer zu erkennen und eindeutig identifizierbar sind und ihre Positionen bekannt sind, kann der Roboter seinen kumulierten Positionsfehler an jeder bekannten Landmarke korrigieren.

Die in der Umgebung verteilten Landmarken bilden zusammen eine topologische Karte. Sie setzt sich aus den Landmarken als Knoten und den Navigationsinformationen zwischen diesen als Kanten zusammen. Da die Karte somit als Graph darstellbar ist, stehen die gängigen Verfahren der Graphenanalyse z.B. zur Pfadsuche, zur Verfügung. Die Idee einer Landmarkenbasierten, topologischen Karte ist die weitestgehende Unabhängigkeit der

genauen Kenntnis der Position, wenn die Navigation zwischen zwei „benachbarten“ Landmarken trivial, d.h. ohne Karteninformationen z.B. nur mit der gegebenen Richtung möglich ist. In dem Fall lässt sich aussreichende, zielgerichtete Navigation über die gesamte topologische Karte durchführen, indem der Roboter sich nur mittels seiner Richtungsinformationen von Landmarke zu Landmarke hangelt. Idealerweise hätte der Roboter Sichtkontakt zu seinem Ziel, da er in dem Fall nicht auf Positions- und Richtungsabschätzung angewiesen ist.

Im hier gestellten Szenario ist die Navigation jedoch uneingeschränkt und nicht frei von Fehlern. So kann die triviale Navigation zwischen zwei Landmarken in der Umgebung scheitern, wenn der Roboter während der Fahrt keine Möglichkeit der Positionskontrolle hat. Es ist also unbedingt erforderlich, zumindest eine ausreichend präzise Schätzung der Position in das Konzept aufzunehmen. Die einfachste Art der Positionsschätzung ist die Odometrie. Mit den dadurch ermittelten Koordinaten kann der Roboter auch zwischen den Landmarken bei Bedarf neu auf das Ziel ausgerichtet werden, wenn er beispielsweise einem Hindernis ausweichen muss.

Für die Planung ist im Allgemeinen die Struktur der topologischen Karte ausreichend. Im Wissensspeicher befinden sich daher die Knoten (Landmarken) und die Information, dass es einen Pfad zwischen zwei Landmarken gibt. Agenten, die mit der Fähigkeit ausgestattet sind, die Landmarken visuell zu erfassen, könnten, unter der Voraussetzung, dass von einer Landmarke aus die jeweils Benachbarten sichtbar sind, bereits mit diesen Informationen navigieren. In dem hier gegebenen Szenario müssen jeder Kante des Graphen noch Navigationsinformationen hinzugefügt werden. Da diese für die Planung im Grunde uninteressant sind und in erster Linie der Ausführung der Pläne dienen, ist dieses Wissen nicht im Wissensspeicher enthalten sondern Teil der Kontrollinformationen des Roboters. Der Agent speichert die Position, an der er die Landmarke entdeckt hat. Erfährt er von einem anderen Agenten, dass dieser eine Landmarke entdeckt hat, nimmt er dessen Information ebenfalls in seinen Wissensspeicher auf. Für einen Plan kann diese „neue“ Landmarke jedoch erst genutzt werden, wenn ihre Position bekannt ist. Hierzu kann der Agent eine Anfrage an andere Agenten schicken, ihm relativ

zu zwei gemeinsam bekannten Landmarken die bislang unbekannte Position einer Landmarke zu schicken. Aus dieser gespeicherten Position kann er während der Planbearbeitung unter Verwendung seiner eigenen geschätzten Position die Richtung und Entfernung des Ziels ermitteln. Da die Verbindugsinformationen zwischen zwei Landmarken normalerweise nicht in die Planung einfliessen, sind gefundene Pfade nur nach Aktionen bzw. Kanten optimal. Muss dagegen bei der Planung der Zeitfaktor mit berücksichtigt werden, ist es erforderlich, diese Informationen mit hinzuzuziehen.

2.5.2 Hinderniskarte

Da nicht jedes Objekt oder Hindernis mit einer Landmarke versehen ist, wird für die systematische Erkundung der Umgebung eine Hinderniskarte benötigt. Diese ist nur für den einzelnen Roboter gedacht und muss nicht austauschbar sein. Auf dieser Karte markiert der Roboter Orte, die er als befahrbar befunden hat und Punkte, an denen seine Abstandssensoren ein Hindernis gefunden haben. Eine solche Karte lässt sich als Rasterkarte umsetzen, in der mit angemessener Auflösung freie, unbekannte und blockierte Gebiete verzeichnet werden. Der Roboter kann auf einer solchen Karte ein unbekanntes Gebiet auswählen, ansteuern und somit überprüfen. Analysen - z.B. der Grösse eines zusammenhängenden, nicht erkundeten Bereichs - sind mit verhältnismässig wenig Aufwand verbunden. Der Nachteil von Rasterkarten ist die fixe Genauigkeit der Karte. Ist die Rasterung zu grob gewählt, verschwinden möglicherweise Details der Umgebung, ist sie zu fein, entsteht unnötiger Aufwand in Datenhaltung und Verarbeitung, z.B. bei der Pfadsuche. Es gibt aber Ansätze, die es ermöglichen, die Vorteile von Rasterkarten zu nutzen, ohne gleichzeitig die Nachteile zu haben. Bekannt ist hier z.B. *Quad Tree*. Bei *Quad Tree* wird die Karte wiederholt in vier Unterbereiche aufgeteilt, bis die gewünschte Präzision erreicht ist. Sind alle vier Verfeinerungsstufen eines Gebiets vom gleichen Typ, z.B. *Blockiert* wird dem gesamten Bereich dieser Wert zugewiesen. Somit erhält man eine dynamische Granulierung der Umgebung bei gleichzeitig hoher Analysierbarkeit der Karte. Im Rahmen dieser Arbeit wurde eine dynamische Karte verwendet, die

in regelmässigen Zeitabständen und abhängig von der bereits klassifizierten näheren Umgebung als „Frei“ markierte Punkte in hoher Präzision setzt. Erkannte Hindernispunkte werden ebenfalls markiert. Da das Setzen neuer Punkte in Abhängigkeit von der Umgebung erfolgt, werden die Punkte an den Rändern der Hindernisse dichter gesetzt. Im Gegensatz zu Rasterkarten ist diese Variante während der Lernphase schneller und robuster. Durch die während der Navigation entstehenden Fehler der Positionsabschätzung können Informationen der Karte teilweise widersprüchlich sein. Die gefundenen freien Punkte werden miteinander verbunden, wobei blockierte Punkte die Verbindungen stören. Auf diese Weise entsteht ein Netz von freien Punkten, mit dessen Hilfe der Roboter Routen planen und navigieren kann. Dieses Konzept würde sogar eine genauere strukturelle Analyse der Umgebung ermöglichen, z.B. Polygonisierung der Hindernisse oder Erzeugung von Raum- und Flurkarten im Falle einer *indoor*-Navigation. Die dynamische Platzierung der Kartenpunkte ermöglicht es auch, Strukturen und Passagen zu finden, die auf einer anderen Karte möglicherweise verschwinden würden. Ohne eine strukturelle Analyse eignet sie sich jedoch nur eingeschränkt für die systematische Erkundung, da es umständlich ist, Bereiche zu erkennen, die noch nicht befahren wurden. Eine Möglichkeit, dennoch mit dieser Karte zu arbeiten, ist, sie zu rastern und somit die Möglichkeiten der Rasterkarten zu nutzen.

2.5.3 Erkundung

Da die Agenten beim Start der Simulation nichts von ihrer Umgebung wissen, müssen sie sich diese Informationen zunächst aneignen. Hierzu können verschiedene Erkundungsstrategien genutzt werden. Für gründliche Aufklärung der gesamten Umgebung eignen sich systematische Erkundungsstrategien, idealerweise in Kombination mit einer Hinderniskarte als Gedächtnis des Verfahrens. Vorausgesetzt es steht genügend Zeit zur Verfügung, werden garantiert alle Landmarken gefunden. Es lassen sich auf diese Weise auch alle direkten Verbindungen jeweils zweier Landmarken finden und somit eine vollständige topologische Karte der Umgebung erzeugen. Es hat sich bei

Versuchen mit jeweils unterschiedlichen Erkundungsstrategien herausgestellt, dass, je nach dem, welche Umgebung vorgegeben wird, die Vollständigkeit der Karte von eher geringer Bedeutung ist. Wesentlich wichtiger ist, dass möglichst viele Probleme in möglichst kurzer Zeit gelöst werden können.

Durch die freie Umgebung ist die Navigation zwischen den Landmarken im Allgemeinen trivial und ziellose, randomisierte Erkundungsstrategien sind im Allgemeinen genauso erfolgreich wie kartenbasierte, systematische Erkundung, dabei aber erheblich weniger aufwändig. Die Landmarken-basierte Karte sowie das Ziel, diese zwischen mehreren Robotern auszutauschen und abzugleichen, macht eine weitgehend tiefenbasierte Erkundung sinnvoll. Diese Erkundungsstrategie lässt die Roboter soweit wie möglich geradeaus durch ihre Umgebung fahren, bevor sie einen zufälligen Richtungswechsel durchführen. Auf diese Weise werden nicht alle Landmarken schnell gefunden, es ist jedoch wahrscheinlicher, dass mehrere Roboter rasch gemeinsame Landmarken kennen und darüber ihre Karten austauschen können. Die Häufigkeit der Richtungswechsel bestimmt die Variante der Erkundung:

- Häufige Richtungswechsel - Gründliche Erkundung eines eingeschränkten Gebiets. Ist mit einem Ziel in der Nähe zu rechnen, bietet sich diese Variante an.

- Seltene Richtungswechsel - Weiträumige Erkundung erleichtert das Auffinden mehrerer verteilter Ziele.

- Keine zufälligen Richtungswechsel - Gut geeignet für weiträumige Erkundung von Labyrinth-artigen Umgebungen.

Ohne zufällige Richtungsänderung kann es eventuell Bereiche der Umgebung geben, die nie erkundet werden. In dieser Arbeit wird ziellose, randomisierte Erkundung mit gelegentlichen Richtungswechseln verwendet. Es ist die effektivste Methode eine freie Umgebung weiträumig zu erkunden und - besonders in Kombination mit einem Multi-Agenten-System - verhältnismässig treffsicher.

Um aus einer Menge von entdeckten Landmarken eine topologische Karte zu konstruieren, sind auch Kanteninformationen erforderlich. Diese Informa-

tionen lassen sich auf verschiedene Weise ermitteln. Eine Möglichkeit ist, beispielsweise, zunächst eine vollständige Vernetzung der Landmarken anzunehmen und durch Befahren zu verifizieren oder zu widerlegen. Die Kanten lassen sich auch durch Analyse einer Hinderniskarte, z.B. über Sichtlinienanalyse finden. Im Rahmen dieser Arbeit wird als Kantenkriterium die direkte Erreichbarkeit einer Landmarke von einer anderen aus genutzt. Eine Kante wird zwischen zwei Knoten genau dann eingefügt, wenn ein Roboter zuvor einen Weg gefahren ist, der ihn von der einen Landmarke zur Anderen geführt hat. Es handelt sich dabei um ein pessimistisches Verfahren, d.h. es wird zunächst angenommen, dass kein Pfad zwischen je zwei Landmarken existiert. Diese Variante unterstützt die Robustheit zielgerichteter Planung, da jede in der Karte vorhandene Verbindung auch tatsächlich existiert und somit ein Plan, der auf diesen Informationen basiert, auch tatsächlich korrekt ist. In einer freien Umgebung kann diese Methode dazu führen, dass zwei Knoten als Nachbarn gelten, obwohl der direkte Pfad zwischen den beiden durch eine dritte Landmarke führt. Auch ist es möglich, dass die Nachbarschaft durch das weiträumige Umfahren eines Hindernisses gefunden wurde. Gerade deshalb ist es sinnvoll, einfache Erkundungsstrategien zu verwenden und somit möglichst einfache Pfade zwischen den Landmarken abzufahren. Der so gefundene Weg ist mit höherer Wahrscheinlichkeit rekonstruierbar und somit besser geeignet.

2.6 Planung

2.6.1 Graphplan

Planung ist neben der Koordination das Herzstück eines erfolgreichen Multi-Agenten-Systems. Aufgabe der Planung ist es, für jeden Agenten einen korrekten Plan zu erzeugen, der wenigstens eines der gestellten Ziele erfüllt und dabei gleichzeitig der Lösung aller gemeinsamen Ziele dient.

Die Verwendung von Robotern als physische Umsetzung der Agenten stellt hohe Anforderungen an das Planungssystem. Die Umgebung eines Roboters ist die „wirkliche“ Welt, die aus Sicht der Künstlichen Intelligenz ei-

ne unzugängliche, nicht-episodische, nicht-deterministische, dynamische und kontinuierliche Umgebung darstellt und somit extrem hohe Anforderungen an das System stellt. Um die Problemstellung zu entschärfen wurden die Umgebungs-Eigenschaften „dynamisch“, „nicht-episodisch“ und „nicht-deterministisch“ durch gezielte Beschränkungen des Versuchsaufbaus herausgenommen. Die Verwendung von abstrakten Aktionen reduziert den Einfluss der kontinuierlichen Umgebung. Die Navigation als Aktion ist jedoch stets kontinuierlich und muss daher entsprechend behandelt werden. Die Unzugänglichkeit der Umgebung erfordert Erkundung, die entweder als eigenständige Phase oder als Teil der Planung ausgeführt sein kann.

Durch entsprechende Formulierung des Szenarios und Gestaltung der Umgebung ist jede Bewegungsaktion der Agenten revidierbar. Somit lässt sich jeder Punkt der Karte garantiert erreichen. Die beiden nicht revidierbaren Aktionen „Feuerlöschen“ und „Retten“ behindern sich nicht gegenseitig und sind von einander unabhängig. Somit gibt es zur Lösung des Problems immer einen Pfad der durch Tiefensuche ohne Backtracking - wenn die jeweils nächste Aktion nicht deterministisch ausgewählt wird - gefunden werden kann. Dieser ist allerdings nicht zwangsläufig auch der optimale Pfad. Um einen optimalen Pfad zu finden, bedarf es eines geeigneten Suchverfahrens, z.B. Breitensuche oder iterative Tiefensuche. In dieser Arbeit wird Graphplan unter der Annahme eingesetzt, dass es sich bei den Gegebenen um „natürliche“ Probleme handelt, für die Graphplan, laut der Entwickler, besonders gut geeignet sein soll. Graphplan wurde bereits im Abschnitt 1.3.6 kurz vorgestellt.

Bedingt durch die Gestaltung des Szenarios sind die Lösungen der Probleme, aus Sicht jedes einzelnen Agenten, ausschliesslich sequentiell. Die Bearbeitung erfolgt immer nach dem Schema:

1. Steuere ein Problem an

2. Bearbeite das Problem

Dieser Ablauf ist für jedes Problem durchzuführen. Da die Probleme selbst nicht von einander abhängig sind, ist die Reihenfolge der Bearbeitung aller

Probleme - unter der Annahme, das stets der kürzeste Pfad zwischen zwei Orten genutzt wird - das einzige Optimalitätskriterium.

Graphplan nutzt zur Steigerung der Effizienz der Suche die im Rahmen der Grapherzeugung gefundenen Abhängigkeiten zwischen *Vorbedingungen*. Es werden nur einfache Abhängigkeiten gefunden, also nur solche, die zwischen genau zwei Informationen bestehen. Somit sind, wenn der Plan die Länge des längsten Plans, der zwei beliebige Ziele erfüllt, überschritten hat, keine Abhängigkeiten mehr im Graph zu finden. Ist der Plan, der alle Ziele erfüllt, länger, steigt der Aufwand der Plansuche deutlich an. Das liegt daran, dass nur jeweils zwei Ziele als nicht gleichzeitig erfüllbar angesehen werden, also von einander abhängig sind. Mit besagtem längsten Pfad zur Erfüllung zweier Ziele ist es im erzeugten Graphen möglich, jeweils einen Plan zu konstruieren, der zwei beliebige Ziele gleichzeitig erfüllt. Daher gibt es keine Abhängigkeiten zwischen den Zielen mehr und die Suche nach dem Plan verhält sich wie eine normale iterative Tiefensuche, jedoch mit dem durch Graphplan erweiterten Suchraum.

Eine weitere Effizienzsteigerung kann durch das Speichern der in jeder Ebene des Graphen unerfüllbaren (Teil-)Zielzustände erreicht werden. Diese Speicherung wird auch verwendet, um unlösbare Probleme erkennen zu können.

Gut geeignet ist Graphplan in diesem Szenario für ein oder zwei Ziele, da hier noch vollständig von den Eigenschaften des Graphen profitiert wird, sowie für die später noch erläuterte zentrale Planung, welche die gleichzeitigen Aktionen mehrerer Agenten berücksichtigt. Allerdings wird in der implementierten Variante von Graphplan - wie auch allgemein üblich - nicht auf die tatsächlichen Pfadkosten Rücksicht genommen. Alle Aktionen haben aus Sicht des Algorithmus' die selben Kosten; somit ist der scheinbar günstigste Pfad also nur der der Anzahl der abgefahrenen Kanten nach kürzeste. Gerade für die Navigation wären die Kosten und somit der zu erwartete Zeitaufwand von besonderem Interesse, da dies Voraussagen über die erwartete Position der Roboter zu bestimmten Zeitpunkten ermöglichen würde. Möchte man das Szenario in Richtung echte Kooperation - z.B. bei nur gemeinschaftlich lösbaren Problemen - erweitern, ist die Zeit ein wichtiger Bestandteil der

Planung und der Koordination. Auch bei der geplanten Verteilung exklusiver Ressourcen sollte der Zeitfaktor berücksichtigt werden. Es ist zwar auch möglich, Planung und Koordination unter Ausserachtlassung der Zeit durchzuführen; es führt dann allerdings mit unter zu Situationen in denen ein Agent untätig bleibt und auf Andere warten muss.

Graphplan bietet durch die Vorverarbeitung des Planungsproblems während der Erstellung des Planungsgraphen einige Möglichkeiten zur Steigerung der Effizienz des Planers. So können Fakten, Aktionen und deren Auswirkungen und Anforderungen auf einen, je nach Umfang des Problems, kleinen Zahlenwert *gemapped* werden, was während der Plansuche selbst die Konstruktion der möglichen Aktionen eines Zeitschritts, das Ueberprüfen der Lösbarkeit und die Erzeugung des Teilziels für den vorherigen Zeitschritt erleichtert.

2.6.2 Massnahmen zur Beschleunigung der Planungsphase

Die Verwendung von Graphplan als Planungseinheit ist für das gegebene Szenario nicht effizient. Wie bereits geschildert liegt dies daran, dass die erzeugten Pläne immer sequentiell sind. Versuche haben gezeigt, dass Graphplan bei mehr als zwei zu erfüllenden Zielen deutlich an Leistungsfähigkeit verliert. Bei hoher Komplexität des zu Grunde liegenden Graphen reichen bereits drei oder vier Ziele aus um die Laufzeit der Planung von wenigen Sekunden auf etliche Minuten zu erhöhen. Daher wurde als auswählbare Alternative ein Planungssystem implementiert, welches die Navigationsplanung von der Gesamtplanung trennt. Das im Folgenden beschriebene Verfahren erzeugt einen sequentiellen, vollständig geordneten Plan für eine beliebige Anzahl Agenten.

Der Graph wird zunächst auf die Ziele und die Startzustände der Agenten reduziert. Die Kanten dieses vereinfachten Graphen werden aus dem günstigsten Pfad zwischen zwei Zielen gebildet und sind mit der Pfadlänge als Kosten gewichtet. Mit diesem gewichteten Planungsgraphen lässt sich nun mittels *Branch-and-Bound Search* die optimale Reihenfolge der Ziele und deren Verteilung auf die Agenten finden. Im Fall der verteilten Planung sind die

Ziele bereits verteilt; es muss nur noch die günstigste Reihenfolge gefunden werden.

Zur Berechnung der optimalen Pfade zwischen je zwei Zielen wird der bereits verfügbare Graphplan hergenommen.

Dieser Alternativplaner ist im Falle hoher Komplexität und mehrerer Ziele effizienter als Graphplan. Dafür ist er allerdings weniger allgemeingültig. Im Gegensatz zu Graphplan ist er nicht mehr einsetzbar, sobald die Ziele von einander abhängen und beispielsweise die Lösung eines Ziels ein anderes unlösbar macht. Wenn nur ein Ziel zu lösen ist ist dieser Ansatz sogar aufwändiger als Graphplan, da zusätzlich zur obligatorischen Planung des Weges zum Ziel noch weitere Arbeitsschritte hinzukommen.

Branch-and-Bound Search ist wie bereits erläutert ein kostenbasiertes Suchverfahren, bei dem immer der günstigste bisherige Pfad expandiert wird.

Um letztlich den optimalen Plan zu erhalten, muss für das Suchverfahren die Definition eines Zustands entsprechend gewählt werden. Wichtig bei der Beschreibung eines Knotens des Suchbaums sind die bis zu diesem Knoten anfallenden Kosten. Im Gegensatz zur Mono-Agenten-Lösung ist der optimale Plan für mehrere Agenten derjenige Plan, der alle Ziele löst und dabei die geringste Tiefe hat. Erst als sekundäres Optimalitätskriterium wird die Gesamtzahl der eingeplanten Aktionen herangezogen.

Jeder Zustand beinhaltet eine Liste der noch nicht eingeplanten Ziele. Die Expansion dieses Zustands ist dann die Menge aller möglichen Zustände, bei denen ein Ziel einem Agenten zugeordnet wurde. Mit entsprechend neu errechneten Kosten werden die so konstruierten Zustände für die weitere Suche genutzt und in die Liste der noch nicht expandierten Knoten einsortiert. Das gewählte Ziel wird aus der Liste der nicht verteilten Ziele entfernt. Wenn diese Liste leer ist wurde ein Plan gefunden, bei dem alle Ziele verteilt sind. Ist dieser gleichzeitig auch noch der günstigste Pfad, wurde somit ein optimaler Plan zur Lösung aller Ziele gefunden.

Der vollständige Planungsablauf sieht wie folgt aus:

1. Ermittle beteiligte Agenten aus dem Ausgangszustand

2. Ermittle beteiligte Landmarken aus den Zielen

3. Für jeden Agenten:

   (a) Berechne die kürzesten Pfade vom Ausgangszustand zu allen Zielen und speichere Pfade und Kosten

   (b) Berechne die kürzesten Pfade von jedem Ziel zu jedem Anderen und speichere Pfade und Kosten

4. Wende *Branch-and-Bound Search* an, um die optimale Verteilung und Reihenfolge der Ziele zu finden

Der grösste Aufwand dieses Verfahrens steckt in der Berechnung der günstigsten Pfade zwischen den Zielen. Diese liessen sich durch heuristische Verfahren erheblich schneller berechnen. Als Heuristik bei der Pfadplanung im zweidimensionalen Raum eignet sich die euklidische Distanz. Dies ist allerdings nur Möglich, wenn die Positionen der Landmarken mit hinreichender Präzision bekannt sind.

Die gefundenen günstigsten Pfade lassen sich zwar speichern und wiederverwenden, bei neu erworbenen Informationen gilt jedoch auch hier - zumindest ohne die Verwendung von Heuristiken -, dass sich ohne die Neuberechnung der Pfade keine Aussage über den Nutzen der Neuplanung treffen lässt.

Bei diesem Planungsverfahren ist - bedingt durch seinen Ablauf - relativ leicht verteilte mit zentraler Planung vergleichbar. Bei der verteilten Planung werden die Ziele vor der Planung verteilt, bei der zentralen Planung erst während der Planungsphase. Daraus ergibt sich, dass in der zentralen Planung für jeden Agenten die günstigsten Pfade zwischen allen Zielen, in der Verteilten dagegen nur zwischen den Zugewiesenen berechnet werden müssen. Bei der Erzeugung des eigentlichen Plans ist im zentralen Fall die Tiefe des Suchbaums durch die Anzahl aller Ziele festgelegt, im verteilten Fall ist sie nur die Anzahl der zugewiesenen Ziele. Bei zentraler Planung ist der Verzweigungsfaktor des Suchbaums in jeder Ebene die Kombination aller noch nicht eingeplanten Ziele mit den Agenten, die sie bearbeiten können; im verteilten Fall entspricht er der Anzahl noch nicht verteilter Ziele. Bei zentraler Planung existieren für alle Agenten und Ziele bis zu $a^n * n!$ mögliche

Lösungspfade; n entspricht dabei der Anzahl an Zielen, a ist die Anzahl der Agenten. Im verteilten Fall müssen je nach Aufteilbarkeit der Ziele zwischen $a*(n/a)!$ - wenn die Ziele gleichmässig verteilt sind - und $n!$ - wenn alle Ziele auf einen Agenten fallen - Pläne überprüft werden.

2.6.3 Nebenläufigkeit

Während der Entwicklung des Steuerprogramms für die Simulation muss die Funktionsweise der Simulation beachtet werden. Mit jeder Zeiteinheit werden die *Controller* jedes aktiven Objekts aufgerufen. Die Simulation führt die abschliessenden Berechnungen einer Zeiteinheit erst dann durch, wenn alle Steuerprogramme die Kontrolle an die Simulation zurückgegeben haben. Im gegebenen Szenario ist zu erwarten, dass die Planung die Dauer einer normalen Zeiteinheit bei weitem übersteigt. Um dennoch eine störungsfreie Echtzeitsimulation zu ermöglichen, gibt es zwei Methoden:

1. Die Planung erfolgt in einem eigenen Thread. Auch wenn das Steuerprogramm die Kontrolle zurückgibt, läuft dieser weiter. Ist die Arbeit abgeschlossen, kann das Ergebnis ausgelesen und genutzt werden. Vorteil: Echte Nebenläufigkeit. Problem: Zusätzliche Threads verfälschen die jedem Agenten zur Verfügung stehende Rechenkapazität während eines Simulationszeitschritts.

2. Die Planung ist Teil des Zustands des Agenten. Sie wird nach bestimmter Zeit in der Ausführung unterbrochen, der aktuelle Suchfortschritt bleibt erhalten. Der *Controller* gibt die Kontrolle an die Simulation zurück, damit diese den nächsten Zeitschritt berechnen kann. Beim nächsten Aufruf wird die Suche an der selben Stelle fortgesetzt. Vorteil: Jedem Agenten kann eine bestimmte Rechenkapazität zugeteilt werden. Ändert sich an der Situation, z.B. an dem Wissen des Agenten, etwas, kann die Planung jederzeit abgebrochen werden. Problem: Durch die Unterbrechung verlängert sich die ohnehin schon erhebliche Laufzeit der Planung.

Ein Problem der nebenläufigen Planung ist, dass - bedingt durch das gegebene Szenario - mit häufigen Änderungen des Wissens der Agenten zu rechnen ist. Läuft die Simulation während der Planung weiter, ist nur in wenigen Fällen - zumeist bei entweder sofort lösbaren oder unlösbaren Problemen - mit der erfolgreichen Terminierung des Planungsalgorithmus zu rechnen. Sobald die Situation komplexer wird besteht die Gefahr, dass die Planung durch neue Informationen ständig unterbrochen wird und nicht zu einem Ergebnis kommt.

Daher wurde beschlossen, die Planung nicht nebenläufig durchzuführen. Für die Simulation bedeutet dies, dass die Planung scheinbar innerhalb eines Zeitschritts erfolgt, obwohl tatsächlich mehr Zeit vergangen ist. Nicht nebenläufige Planung ist zwar aus Sicht der Simulation die bessere Wahl, da Vergleiche und Beobachtungen einfacher werden, für eine praktische Umsetzung ist sie allerdings nicht geeignet: Es würde nämlich bedeuten, dass alle Agenten solange stillstehen und warten müssen, bis auch der Letzte seine Planung abgeschlossen hat.

2.6.4 Unbekanntes Wissen

Der Planungsalgorithmus muss mit der erschwerenden Bedingung zurecht kommen, dass Teile des Wissens über die Umgebung unbekannt sind. Dabei kann zwischen drei Varianten unterschieden werden.

1. Unbekanntes irrelevantes Wissen: Wissen, das auch wenn es dem Agenten zur Verfügung stünde, keinen Einfluss auf seinen Plan hätte.

2. Unbekanntes lösungsrelevantes Wissen: Wissen, das über die Lösbarkeit eines Problems entscheidet.

3. Unbekanntes effizienzrelevantes Wissen: Wissen, dessen Verfügbarkeit den Plan verbessern kann.

Ein Problem des Umgangs mit neu erworbenem, vorher unbekanntem Wissen ist, dass nicht zwischen diesen drei Varianten unterschieden werden kann ehe es in die Planung einbezogen worden ist.

Bei dynamischem Planen spricht man von einem robusten Plan, wenn Veränderungen der Umgebung oder neu erworbenes - also bislang unsicheres, nun validiertes - Wissen die Ausführbarkeit des Plans nicht einschränkt. In dem gegebenen Szenario ist der globale, statische Plan nach dieser Definition robust, da entweder gilt, dass mit dem unvollständigen Wissen kein Plan erzeugt werden konnte, oder dass der Plan nach wie vor abgearbeitet werden kann. Dies ist jedoch nur wahr, da in dem gegebenen Szenario neues Wissen einen gültigen Plan nicht invalidieren kann. Es ist jedoch möglich, dass mit neu erworbenem Wissen ein besserer Plan erzeugt werden kann. Leider lässt sich vor der Berechnung des neuen Plans nicht feststellen, ob er besser sein wird als der Bisherige. Es ist daher abzuwägen, ob der Aufwand, einen komplett neuen Plan zu erzeugen, durch eine eventuell zu erwartende Verbesserung gerechtfertigt ist. In dieser Arbeit wird die Planung bei Erwerb neuen Wissens stets erneut durchgeführt. Da neue Informationen oftmals weitere Veränderungen des Wissens verschiedener Agenten mit sich bringen können, wird vor dem Beginn der Planung auf das Ende des Informationsaustauschs gewartet.

2.7 Koordination

Die Koordination der einzelnen Pläne ist der wesentlichste Bestandteil dieser Arbeit. Die Koordination dient in erster Linie der Erzeugung eines globalen, korrekten und vollständigen teilweise geordneten Plans. Eine weitere Aufgabe der Koordination ist die Vermeidung von Konflikten in der Zielverteilung, -auswahl und -bearbeitung. Und schliesslich dient sie der schrittweisen Verbesserung des globalen Plans durch Austausch und Umsortierung von Aufgaben sowie der Kooperation bei der Lösung einzelner Probleme.

In dem hier vorgestellten Szenario ist Kooperation nicht zur Lösung eines einzelnen Problems erforderlich und wird deswegen nicht weiter betrachtet.

Der Ablauf der koordinierten Planerzeugung der Agenten folgt dem folgenen Schema:

1. Neue Informationen invalidieren den aktuellen Plan und stossen eine

Neuplanung an

2. Der Agent wird in seiner Bewegung gestoppt und wartet kurz auf gegebenenfalls weiteres aus den neuen Informationen konstruiertes Wissen
3. Der Agent bestimmt die Menge aller noch offenen Ziele
4. Der Agent führt für jedes Ziel einen Lösbarkeitstest aus und verwirft alle nicht lösbaren Ziele
5. Der Agent sendet Koordinationsinformationen an alle anderen Agenten. Diese beinhalten jeweils ein Ziel und eine Schätzung des Aufwands zu dessen Erfüllung
6. Der Agent wartet eine Weile auf Koordinationsnachrichten der anderen Agenten; Ist der erwartete Aufwand für die Bearbeitung eines Ziels bei einem anderen Agenten geringer, entfernt der Agent das Ziel aus der Menge seiner Ziele
7. Der Agent erzeugt einen Plan, der alle seine Ziele beinhaltet
8. Der Agent beginnt mit der Abarbeitung seines Plans

Koordination beginnt bereits vor der Erzeugung der einzelnen Pläne mit der Verteilung der Aufgaben. Es ist einleuchtend, dass diese Verteilung ohne Übersicht über globale Bedingungen nur aufgrund lokaler Informationen erfolgen kann. In dieser Arbeit wird als Grundlage der Verteilung der einzelnen Aufgaben ein Plan erzeugt, der nur das jeweils zu untersuchende Ziel berücksichtigt. Dieser Plan dient als Lösbarkeitstest und ermöglicht es, den Aufwand der Bearbeitung des jeweiligen Ziels abzuschätzen. Da der verwendete Planungsalgorithmus „vollständig“ ist, lassen sich so für jeden Agenten Probleme bestimmen, die dieser nicht bearbeiten kann. Für die weitere Bearbeitung müssen diese nicht lösbaren Ziele nicht weiter betrachtet werden.

In dem gegebenen Szenario gibt es Agenten mit spezialisierten Expertisen. Beispielsweise gibt es 2 Agenten, die in der Lage sind, Feuer zu löschen und zwei weitere, die Verletzte versorgen können. Die Erkundung hat ergeben: es sind fünf Probleme zu lösen: Drei Brände und zwei Verletzte. Ist allen

Agenten die selbe Karte gegeben und sucht jeder Agent für jedes Problem einen Plan, so werden bei einem Feuerlösch-Agenten die Verletzten, bei einem Sanitäter hingegen die Feuer keinen validen Plan erzeugen. Diese nicht bearbeitbaren Probleme werden entfernt. Entsprechend gilt beispielsweise, wenn einem der Feuerlösch-Agenten die Position eines Brandes nicht bekannt, ist auch dieses Problem nicht bearbeitbar und wird entfernt.

Im nächsten Schritt verständigen sich die Agenten untereinander, wer welches Problem übernimmt. Gibt es nur einen Agenten, der ein Problem bearbeiten kann übernimmt er es automatisch. Können mehrere Agenten das selbe Problem behandeln, gibt es verschiedene Möglichkeiten, zu entscheiden, wer dieses Problem übernimmt. Im einfachsten Fall übernimmt derjenige Agent eine Aufgabe, der für ihre Bearbeitung den geringsten Aufand hat. Aus dem als Lösbarkeitstest erzeugten Plan lässt sich der lokale Aufwand eines Problems in Aktionen herauslesen. Es muss allerdings auch beachtet werden, dass nicht ein Agent alle Aufgaben übernimmt, nur weil er zufällig für jede einzelne den geringsten Aufwand hat. Zur Verteilung der Ziele gibt es verschiedene Ansätze, die beispielsweise einem Ziel neben dem Nutzen für das Gesamtsystem auch einen „persönlichen“ Wert zuweist, den nur der jeweilige Agent kennt. Auf Basis dieser Bewertung der einzelnen Ziele lassen sich mittels Verhandlungen oder Auktionen Aufgabenverteilungen erzielen, die die Gesamteffizients des Systems positiv beeinflussen. Effizienter, jedoch auch aufwändiger wäre es, für jeden einzelnen Agenten zunächst einen Plan mit allen lösbaren Zielen zu erzeugen und erst in einem späteren Schritt eventuell auftretende Konflikte zu beheben. So könnte beispielsweise derjenige Agent ein Ziel aus seinem Plan entfernen, bei dem der Gesamtaufwand mit diesem Ziel am höchsten ist. Es könnte auch nur der Agent das Ziel behalten, der es in seinem Plan am schnellsten erfüllt. Jede Veränderung der Pläne zerstört die Optimalität. Da im gegebenen Szenario die Bearbeitung der Ziele im Grunde unabhängig von einander erfolgt, lassen sich einzelne Ziele relativ einfach aus dem Plan entfernen; im Allgemeinen sind jedoch weitere Bearbeitungen erforderlich, um die Korrektheit des neuen Plans sicherzustellen.

Für diese Arbeit wurde eine einfache Variante gewählt, bei der die Auf-

gaben nur nach der lokalen Aufwandsschätzung verteilt werden. Dieses recht einfache Verfahren wird in diesem Fall durch die unbekannte Umgebung gerechtfertigt. Im Normalfall werden häufig neue Informationen gefunden und die Pläne damit ungültig, wodurch die gesamte Planung - inklusive Zielverteilung und Koordination - erneut durchgeführt werden muss. Eine beschleunigte Planerzeugung ist in diesem Fall daher der möglichen Effizienzsteigerung vorzuziehen.

Im oben geschilderten Beispiel würde jeder Agent jedes Ziel mit einer Bewertung versehen, die der Länge des Planes zur Erfüllung des Ziels in Aktionen entspricht. Diese Information sendet jeder Agent an jeden Anderen und nur derjenige Agent behält das Problem, bei dem dieser Wert am geringsten ist.

Da es vorkommen kann, dass zwei Agenten zur Lösung einer Aufgabe die gleiche Anzahl an Aktionen benötigen wird als weiteres, willkürliches Verteilungskriterium die Kommunikations-Adresse des Agenten genutzt. Auch hier gibt es viele mögliche Ansätze die über die Betrachtung des Problems selbst hinausgehen. In diesem Szenario bieten sich jedoch solche schnellen Lösungen an.

Sind die Ziele verteilt, erzeugt jeder Agent für sich einen Plan in dem alle seine Aufgaben berücksichtigt sind. Dieser Plan ist mit den gegebenen Informationen für seine Ziele optimal und valide. Entsprechend sind nach der Erzeugung der Einzelpläne alle lösbaren Aufgaben eingeplant. Der Plan reicht daher, so wie er nun vorliegt, bereits aus, um alle zu diesem Zeitpunkt lösbaren Probleme zu bearbeiten. Dies gilt jedoch nur aufgrund der Unabhängigkeit der Einzelziele voneinander; sind alle Teilziele einzeln erfüllbar, existiert auch ein Plan, der alle Ziele beinhaltet. Bestehen Abhängigkeiten oder schliessen sich Ziele gegenseitig aus, ist zusätzlicher Koordinationsaufwand notwendig, um Korrektheit und Vollständigkeit zu erreichen.

Da die Teilpläne bereits zusammengenommen einen gültigen Gesamtplan bilden, dient jede weitere Verbesserung der Koordination nur der Effizienzsteigerung. Es stellt sich hierbei, insbesondere auf Grund des durch Erkundung ständig veränderte Umgebungswissens, die Frage, ob die erzielbare Steigerung der Bearbeitungsgeschwindigkeit, vor allem jedoch die tatsächliche

Nutzbarkeit dieser Steigerung, den Aufwand einer Verbesserung des globalen Plans rechtfertigt.

Gerade in der frühen Phase eines Simulationslaufs werden so regelmässig neue Informationen über die Umgebung erworben, dass selten ein Plan tatsächlich durchgeführt werden kann. Es zeigt sich, dass, von „Eine-Aktion-Plänen“ abgesehen, deren Bearbeitung geschützt ist, in den seltensten Fällen erzeugte Pläne bearbeitet werden können, bevor sie ungültig werden. Da hierdurch erheblicher Mehraufwand in der Planung entsteht, wäre es eine Mögliche Effizienzsteigerung, nicht weiter als die am schnellsten bearbeitbaren Probleme vorauszuplanen. Hierdurch geht die Chance näher an Optimalität zu kommmen verloren.

2.8 Zentrale Lösung

Um eine Bewertung der verteilten Planung zu ermöglichen, wird zum Vergleich ein zentraler Plan erzeugt. Da der zentralen Planungsstelle alle Informationen aller Agenten zur Verfügung stehen ist der so erzeugte Plan korrekt und optimal. Vor der Erzeugung dieses Plans wird jedes Teilproblem, wie bei den einzelnen Agenten auch, auf Lösbarkeit überprüft. Auch hier gilt: Jedes Teilproblem für das es einen Plan gibt, lässt sich auch in einem Gesamtplan lösen.

Für die zentrale Planung eignet sich Graphplan noch besser, da hier die Paralellisierung der Ausführung und die Verteilung auf mehrere Agenten durch den Graphen unterstüzt werden. Je stärker die Probleme auf die Agenten verteilt werden können, desto schneller lässt sich ein Plan finden. Gleichzeitig bedeutet das jedoch auch, dass immer alle Agenten in den zentralen Plan eingebunden sind was die Komplexität des Problems und damit auch die Planungsdauer erhöt.

Um an alle Informationen zu gelangen, die den Agenten zur Verfügung stehen, hört der Zentrale Planer die Komunikation der Agenten mit und erzeugt sich so sein Wissen. Er nimmt selbst nur zur übermittlung des berechneten Plans an der Kommunikation teil.

In der Webots-Simulation übernimmt der Supervisor die Rolle des zentra-

len Planers. Die ihm zur Verfügung stehenden Informationen können neben dem Planen auch für Visualisierung genutzt werden. Der Supervisor ist in Webots die Einzige Entität, die beliebige Aenderungen an der Simulations-Umgebung vornehmen kann. Im gegebenen Szenario stellt er Beispielsweise die Probleme, die an den Landmarken bestehen farblich da. Ist ein Problem bearbeitet ändert der Supervisor die Farbe der entsprechenden Landmarke. Sind alle Probleme gelöst sendet der Supervisor eine „Stop“ Nachricht an alle Agenten, die bei Erhalt dieser Nachricht jede weitere Handlung einstellen. Zur Auswertung senden die Agenten noch ihre gemessenen Zeiten an den Supervisor, die dieser entsprechend Verarbeitet und Ausgibt.

Kapitel 3

Ergebnisse

3.1 Beobachtungen

3.1.1 Bearbeitung aller Ziele

Bei der gewählten Erkundundungsstrategie und durch das Ausserachtlassen von Beschränkungen der Zeit oder ähnlicher Ressourcen ist das Bearbeiten aller gestellten Ziele garantiert. Bis die Karten einiger Agenten ausgetauscht werden, erfüllen die Agenten lösbare Aufgaben sobald sie sie finden. Sind die Karten schliesslich abgeglichen, steuert der entsprechende Agent die für ihn neu lösbaren Ziele an und bearbeitet sie. Wie bereits geschildert, ist oftmals damit zu rechnen, dass während der Zeit der Abarbeitung des Plans neue Informationen verfügbar werden und der Plan damit invalide wird. Er wird mit vollem Aufwand neu berechnet, bleibt in den meisten Fällen gegenüber dem alten Plan jedoch unverändert. Dieser Mehrfachaufwand der Planung ist das grösste Effizienzproblem bei der Bearbeitung der Probleme. Der Grund hierfür ist der sehr problematische Umgang mit unbekanntem und neu erworbenem Wissen. Um hier Abhilfe zu schaffen, müsste ein Verfahren entwickelt werden, um den Nutzen der Neuplanung vor der eigentlichen Neuplanung zu ermitteln. Interessant wäre in diesem Zusammenhang ein Vergleich zwischen ungeplantem „Finden und Bearbeiten“ in einem Ameisen- oder Schwarmsystem und der geplanten, zielgerichteten Bearbeitung der einzelnen Ziele in einem MAS.

Bedingt durch die randomisierte Erkundungsstrategie wird dasselbe Szenario nicht immer auf die selbe Weise bearbeitet. Je nach dem, wie schnell und von wem die Probleme entdeckt werden, können Planungsaufwand und Neuplanungshäufigkeit stark variieren. Es kann auch vorkommen, dass einzelne Probleme erst nach längerer Suche gefunden und somit bearbeitbar werden.

3.1.2 Vergleich Verteilter und Zentraler Planung

Ein Vergleich der verteilten mit zentraler Planung ist nicht ohne Weiteres möglich. Zum Einen ist die Qualität der berechneten Pläne zu vergleichen, zum Anderen die Geschwindigkeit der jeweiligen Planerzeugung. Um diesen Vergleich zu ermöglichen, wurde ein festes Szenario gestaltet in dem von jedem der drei Agententypen (Feuerwehr, Sanitäter und Kundschafter) je zwei Verwendung finden. Diese starten an verschiedenen Orten, die möglichst gleichmässig auf der ganzen Karte verteilt sind. In der Umgebung sind 20 Landmarken positioniert, die zumeist Teile von Hindernissen und Durchgänge markieren. 12 dieser Landmarken sind mit Problembeschreibungen versehen; jeweils sechs sind Brände und Verletzte. Der Vergleich der Planungsmethoden erfolgt über die jeweils benötigte Zeit zur Bearbeitung aller gestellten Probleme. Diese Zeit beinhaltet Erkundung, Planung, Koordination und Planbearbeitung. Während die Erkundung und die Planabarbeitung durch die Laufzeit der Simulation abgedeckt sind, ist die Planungszeit, die den Stillstand der Simulation erfordert, nicht so einfach ermittelbar. Eine schlichte Messung der Echtzeit ist ungeeignet, da zeitgleich laufende Prozesse die Ergebnisse verfälschen. Die Rechenzeit der Planung ist nicht leicht zu ermitteln, da die Steuerprogramme der Agenten, deren Prozesszeit zu diesem Zweck ausgelesen werden müsste, als Threads des Webot-Prozesses umgesetzt sind. Somit ist mit ANSI-C++ nur die Prozesszeit des Hauptprozesses - also von Webots - auslesbar. Bei zentraler Planung wird durch den Stillstand der Simulation die für die Planung eingesetzte Prozesszeit auch nahezu vollständig für die Planung genutzt. Im verteilten Fall werden die ausgelesenen Zeiten jedoch unvorhersagbar verfälscht, da die einzelnen planenden

Agenten untereinander um die CPU konkurrieren und somit gemeinsam die Prozesszeit des Hauptprozesses erhöhen. „Unvorhersagbar“ ist das Ausmass dieser Konkurrenz, da Agenten zum Teil parallel zu rechnen Anfangen, teilweise aber auch zu unterschiedlichen Simulationszeiten. Auch die Länge der Planung verfälscht die Messergebnisse. Starten 3 Agenten zeitgleich mit der Planung kann man annehmen, dass jeder ein Drittel der verfügbaren Rechenleistung erhält. Ist der Erste fertig steht jedem der verbleibenden Agenten die Hälfte zur Verfügung; somit erhält man für die selbe genutzte Prozessorzeit niedrigere Messergebnisse. Hat einer der sechs Agenten insgesamt beispielsweise gemessene 100 Sekunden mit Planung verbracht, so lässt sich im Nchhinein nicht sagen, ob er diese Zeit wirklich für die Berechnung benötigt hat, oder nur einen Teil davon. Hat er die ganze Zeit stets gleichzeitig mit den anderen Agenten geplant, sollten sie alle 100 Sekunden Planungszeit haben. Die tatsächliche Rechenzeit jedes Agenten ist dann 16 - 17 Sekunden. Es ist nicht anzunehmen, dass immer alle Agenten zeitgleich starten und vor allem ist nicht damit zu rechnen, dass sie stets gleichlang brauchen, sodass die gemessene Zeit höchstwahrsceinlich nicht 6-mal so gross ist wie die tatsächliche, sondern nur ca. 3 - 4 mal. Dieser Faktor ist von der Erkundung stark beeinflusst, da diese die erforderliche Länge der Pläne festlegt. Reichen kurze Pläne aus, liegt der Faktor nahe an 4, bei langen Plänen verschiebt er sich in Richtung 3. Er kann in manchen Fällen auch unter 3 liegen. Diese Abhängigkeiten und Beeinflussungen machen Aussagen über den Aufwand dieses Ansatzes sehr schwierig. So zeigt beispielsweise der Unterschied zwischen den Planungszeiten eines Kundschafters und denen eines Sanitäters, wie hoch der Anteil der, durch regelmässige Neuplanung hervorgerufenen, Lösbarkeitstests gegenüber der tatsächlichen Planung ist. Da die Kundschafter häufig zeitgleich mit anderen Agenten planen, während die mit anderen Fähigkeiten ausgestatteten oft - zumindest Zeitweise - alleine Planen, ist ein Vergleich schwierig. Die genauen Zeiten der Teilprozesse liessen sich möglicherweise mit Betriebssystem-abhängigen Massnahmen ermitteln.

Ein anderes Problem ist die Verteiltheit der Agenten. Normatlerweise kann man davon ausgehen, dass der zentrale Planer über ein Vielfaches der Rechenleistung verfügt, die ein einzelner Agent besitzt. Auf Grund der oben

geschilderten Probleme lässt sich die Planungszeit jedes einzelnen Agenten nicht bestimmen, die Gesamtzeit hingegen wenigstens abschätzen. Der Einfachheit halber wird die durchschnittliche Planungszeit der Agenten herangezogen um die Gesamtlaufzeit eines Testlaufs zu ermitteln. Vergleicht man diesen Wert mit dem gemessenen Planungsaufwand der zentralen Planung, erhält man einen brauchbaren Vergleich der beiden Ansätze unter der Annahme, dass beide Systeme gleich leistungsstark sind. Teilt man die zentrale Planungszeit durch die Anzahl der Agenten, also durch 6, erhält man den Wert, den die zentrale Planung hätte, wenn sie so leistungsstark wäre wie alle Agenten zusammen.

Die in der Tabelle 3.1 angegebenen Zeiten sind mit den Faktoren 3 und 4 berechnet. Der tatsächliche Zeitaufwand liegt meistens zwischen diesen Werten. Der tatsächliche Zeitaufwand eines Testlaufs entspricht in etwa der Summe aus der gesamten Planungszeit und der Simulationszeit. Möchte man daraus die Zeit bei „echter“ verteilter Berechnung abschätzen, so muss man berücksichtigen, dass sich diese nach dem Agenten mit dem grössten Planungsaufwand richtet. Sie entspricht in etwa der Zeit der Simulation und dem 1,5 - 2-fachen des durchschnittlichen Planungsaufwands. Auch dieser Wert ist empirisch ermittelt und kann nur eine grobe Abschätzung der Zeit liefern.

Die Tabelle 3.1 zeigt deutlich, dass, ungeachtet ob nun ein Faktor von 3 oder 4 angenommen wird, die durchschnittliche Planungszeit immer unter der Simulationszeit liegt. Es lässt sich aus der Varianz der Simulationszeiten auch herauslesen, dass für die aktuelle Situation ein systematisches Erkundungsverfahren, welches die Simulationszeit verkürzt und gleichzeitig stabilisiert, die beste Optimierungsmassnahme wäre. Da die Planungszeit deutlich kürzer ist als die Simulationszeit, liesse sich die Planung - zumindest Teilweise - unmerklich im Hintergrund ausführen, so dass die Simulation und die Bewegung der Agenten fortlaufen kann. Dabei müssen die bereits erläuterten Probleme der Nebenläufigkeit der Planung beachtet werden.

Die Auswertung der zentralen Planung ist erheblich einfacher: Die Gesamtzeit ist die Summe aus Simulationszeit und der um den Leistungsfaktor der planenden Einheit korrigierten Planungszeit. Die Tabelle 3.2 gibt die

S	P/D (F=3)	P/D (F=4)	G (F=3)	G(F=4)
294	176/29	132/22	323	316
426	137/23	103/17	449	443
391	248/41	186/31	432	422
265	193/32	145/24	297	289
421	135/23	101/17	444	438
299	225/38	168/28	337	327
293	377/62	283/47	355	340
464	242/40	182/30	504	494
253	138/23	103/17	276	270
213	192/32	145/24	245	237
472	197/33	148/25	505	497
468	187/31	140/23	499	491
209	173/29	130/22	238	231
273	222/37	167/28	310	301
246	488/81	367/61	327	307
304	220/37	165/28	341	332
264	237/40	178/30	304	294
413	178/30	134/22	443	435
288	143/24	107/18	312	306
225	174/29	131/22	254	244
324	214/36	161/27	360	351

Tabelle 3.1: Zeiten in 20 Versuchsläufen. Verteilte Planung. **S**imulationszeit, **P**lanungszeit/**D**urchschnitt je Agent, **G**esamtzeit(Errechnet)

gemessenen Zeiten mit Leistungsfaktor von 1, 6 und 10 an.

S	P (L=1)	P (L=6)	P (L=10)	G (L=1)	G (L=6)	G (L=10)
165	378	63	38	543	228	193
557	848	141	85	1405	698	642
141	532	89	53	673	230	194
226	608	101	61	834	327	287
281	409	68	41	690	349	322
604	262	44	26	866	648	630
408	459	77	46	867	485	454
164	293	49	29	457	213	193
411	691	115	41	1102	526	452
127	440	73	44	567	200	171
317	281	47	28	598	364	345
114	425	71	43	539	185	157
137	722	120	72	859	257	209
152	762	127	76	914	279	228
318	520	87	52	838	405	370
222	527	88	53	749	310	275
146	302	50	30	448	196	176
291	640	107	64	931	398	355
213	461	77	46	674	290	259
251	604	101	60	855	352	311
262	508	85	51	770	347	313

Tabelle 3.2: Zeiten in 20 Versuchsläufen: Zentrale Planung mit Leistungsfaktor **L**; **S**imulationszeit, **P**lanungszeit, **G**esamtzeit

Vergleicht man die erreichten Zeiten, so erkennt man, dass auf dem Testgerät die verteilte Planung im Schnitt deutlich schneller ist als die Zentrale. Wird die Leistung um die Anzahl der Agenten korrigiert, bleibt die verteilte Planung immer noch im Vorteil. Erst wenn die zentrale Planung auf einer Rechenmaschine läuft, die mindestens 14 mal schneller ist als ein einzelner Agent, ist die zentrale Planung der verteilten vorzuziehen. Es fällt jedoch auf, dass die zentrale Planung im Durchschnitt kürzere Simulationslaufzeiten hat. Abgesehen von der hohen Varianz der randomisierten Aufklärung liegt dies hauptsächlich an den Wartezeiten, die die Agenten bei der verteilten Planung haben, wenn der Lösbarkeitstest durchgeführt wird und auf Koordinationsnachrichten gewartet werden muss. Ein weiterer Grund ist die Qualität der

gefundenen Pläne. Über diese lässt sich jedoch nichts aussagen ohne die verteilt und zentral ermittelten Pläne unter identischen Bedingungen zu vergleichen. Da die Gesamtlaufzeit im verteilten Fall mit dem Leistungsfaktor 6 mit der zentralen Planung mithalten kann, kann man zumindest feststellen, dass, selbst wenn die Verteilung der Aufgaben und die Qualität der gefundenen Pläne nicht optimal sind, die deutlich kürzere Planugszeit diesen Ansatz zu einer äusserst nützlichen Alternative macht.

3.2 Ausblick und Verbesserungsvorschläge

3.2.1 Allgemeine Anmerkungen

Anhand der Beobachtungen und Auswertungen der Simulationsläufe lassen sich relativ leicht eine ganze Reihe von Verbesserungen und Erweiterungen sowohl des Verhaltens der Agenten als auch am gestellten Szenario selbst finden. So ist beispielsweise das Verhalten bei der Kollisionsvermeidung durch Anwendung des Braitenberg-Algorithmus' rein reaktiv. Muss ein Agent also einem Hindernis ausweichen, nimmt er dabei keine Rücksicht auf sein eigenes Ziel. Die Navigation anhand der topologischen Karte ist meistens nicht davon betroffen. Es kann jedoch vorkommen, dass der Agent in die „falsche“ Richtung abdreht und einen grossen Umweg fahren muss, um zu seinem Ziel zu gelangen. Aehnliche eher kleine Verhaltensverbesserungen lassen sich auch in anderen Situationen finden. Von grösserer Bedeutung sind Effizienz steigernde Massnahmen, z.B. bezüglich der Planungsverfahren. Im folgenden werde ich einige Vorschläge zu Verbesserungen und Variationen sowohl der Planungsverfahren als auch des Szenarios aufzählen:

3.2.2 Heuristische Suchverfahren

Das in dieser Arbeit entwickelte Multi-Agenten-System ist für die praktische Umsetzung auf dem Khepera-II-Roboter ungeeignet, da die Planung einen zu hohen Rechenaufwand erfordert. Um dennoch eine Umsetzung zu ermöglichen, ist es erforderlich, ein Konzept zu finden, welches schnell und

einfach qualitativ ausreichende Pläne erzeugt. Zunächst scheint es angebracht, Graphplan gegen ein heuristisches Suchverfahren auszutauschen, das normalerweise deutlich schneller sein sollte. Ausserdem sollte die hier gewählte symbolische Wissensspeicherung überdacht werden, da viele Berechnungen zusätzliche Umwandlungen des Wissens erfordern. So liesse sich die topologische Karte statt als Teil des Weltzustands als eigene Repräsentation beispielsweise mit Heuristiken und Kosten sowohl Speicher- als auch Verarbeitungsfreundlicher darstellen.

Es stellt sich ausserdem die Frage, ob „vollständige Planung“ überhaupt erforderlich ist. Es wäre höchstwahrscheinlich deutlich effizienter, vorab eine gezielte Auswahl einzelner Teilziele vorzunehmen. Hierzu liesse sich eine Heuristik über die Lösung jeweils eines (Teil-)Problems aufstellen - also wie aufwändig die Lösung eines Problems von einem gegebenen Zustand aus ist und wie gross ihr voraussichtlicher Nutzen sein wird. Hierzu liesse sich beispielsweise die Distanz zwischen dem derzeitigen Aufenthaltsort eines Agenten und dem Ort des jeweiligen (Teil-)Problems verwenden.

3.2.3 Planung der Zeit

Um ein Roboter-Multi-Agenten-System im realistischen Fall koordinieren zu können, ist die Berücksichtigung des Faktors „Zeit“ bei der Planung unabdingbar. Der Grund hierfür ist, dass andernfalls Agenten nur durch „Warteaktionen“ synchronisiert werden können. Müssen zwei Agenten zur Erfüllung ihrer Aufgaben gleichzeitig am selben Ort sein würde der Plan des einen Agenten also enthalten: Gehe an den gewünschten Ort; wenn der andere Agent schon da ist, mache weiter, wenn nicht warte. Entsprechendes würde sich auch im Plan des Anderen wieder finden. Diese Art der Synchronisation ist erstens problematisch, da die Gefahr von Deadlocks besteht und zweitens uneffektiv, da der Agent während der nicht vorhersagbaren Wartezeit untätig ist.

3.2.4 Planung mit exklusiven Ressourcen

Ein der Zeitplanung verwandtes Problem ist die Planung exklusiver Ressourcen. Diese müssen entsprechend auf die Agenten verteilt werden. In einem zentralen Plan ist dies durchaus möglich; bei verteilter Planung ergeben sich dagegen erhebliche Schwierigkeiten. Die Ressourcen lassen sich am Besten durch Verhandlung und Scheduling planen. Ermöglichen bestimmte Ressourcen bestimmte Aktionen, wie es bei Werkzeugen der Fall ist, muss festgelegt werden, wer wann welche Ressource bekommt, wie lange er sie haben darf und wie die Übergabe zu einem anderen Agenten erfolgt. Auch die weitere Vorgehensweise, wenn der Agent im Besitz des Werkzeugs ist, ist zu klären. Soll der Agent zunächst alle Arbeiten ausführen für die er das Werkzeug braucht und es dann wieder freigeben oder besser die optimale Handlungsabfolge wählen und das Werkzeug gegebenenfalls länger in Anspruch nehmen? Der Transport eines Werkzeugs ist ebenfalls ein erhebliches Problem, da der Agent, der es braucht, möglicherweise noch andere Arbeiten zu verrichten hat. Das Werkzeug zu holen wäre dann eine zusätzliche Belastung. Agenten könnten hierzu in die Lage versetzt werden, selbst Aufgaben formulieren zu können. Die Komplexität eines Szenarios lässt sich fast beliebig weiter steigern. Wird die Umgebung erst dynamisch und nicht-deterministisch, lässt sich mit Planung und Scheduling nicht mehr viel erreichen. Das sieht man sehr deutlich wenn man betrachtet, wie Menschen in Gruppen Probleme lösen. Viel ist reaktive Massnahme wie beispielsweise „Wenn das benötigte Werkzeug frei ist, nehme ich es mir“, oft aber auch Anfragen und Verhandlung. Der Mensch hat in solchen Fällen den Vorteil, dass er viele Situationen abschätzen kann und über erlernte Strategien und flexibles Umplanen erheblich leichter zu einer gemeinsamen Lösung kommen kann.

3.2.5 Einschränkung der Kommunikationsmittel

Um das Szenario realistischer zu gestalten, liesse sich die Reichweite der Funkkommunikation beschränken. In dem Fall wäre Informationsaustausch und Koordination erheblich erschwert. Die Agenten müssten Kommunikationsnetze aufbauen und Nachrichten weiterleiten. Somit wäre die Bildung

dynamischer Gruppen zur Lösung lokal zusammengehöriger Probleme ein möglicher Ansatz. Auch die Benennung von Boten, also Agenten, die nur Nachrichten transportieren, oder Zentralen, die sich um die Reduzierung der Informationsmenge und um die Koordination örtlich getrennter Agenten kümmern, wäre möglich. In solchen Szenarien kommt es noch weniger auf die Qualität der Lösung an, also wie nah sie an der optimalen Lösung ist. Viel wichtiger ist hier, dass überhaupt alle Probleme auch bearbeitet werden und die Agenten sich nicht gegenseitig behindern.

3.2.6 Systematische verteilte Erkundung

Die Erkundung ist ein wesentlicher Bestandteil dieses Szenarios. In dem hier vorgestellten Ansatz wird sie randomisiert durchgeführt. Das bedeutet auf der einen Seite zwar, dass garantiert jeder Ort der Umgebung irgendwann von einem Agenten besucht wird; es heisst aber gleichzeitig auch, dass Orte unnötigerweise mehrfach angesteuert werden und das Erkunden anderer Gebiete dadurch unnötig spät erfolgt. Durch die Verwendung einer - notfalls auch sehr unpräzisen - Karte liesse sich die Suche systematisieren. Die gesamte Umgebung wird hierzu in Quadranten eingeteilt, die nach einem bestimmten Schema abgefahren werden. Lassen sich die topologischen Karten mehrerer Agenten verbinden, so lassen sich auch diese Erkundungskarten zusammenführen und die Erkundung verteilt weiterführen. Wurden alle Quadranten erkundet, sind somit auch alle Landmarken bekannt. Die Verbindungen zwischen diesen lassen sich dann gezielt vervollständigen.

Literaturverzeichnis

[1] Will Briggs and Diane Cook. Scalable modularity to reduce communication in multiagent planning.

[2] Jeffrey S. Cox and Edmund H. Durfee. Exploiting synergy while maintaining agent autonomy.

[3] M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. A resource logic for multi-agent plan merging, 2001.

[4] M. de Weerdt, F. de Boer, W. van der Hoek, and J. Ch. Specifying uncertainty for mobile robots, 1998.

[5] K. Decker and V. R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 67–82, Hidden Valley, Pennsylvania, 1993.

[6] Keith Decker and Victor Lesser. Designing a family of coordination algorithms. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 73–80, San Francisco, CA, USA, 1995. The MIT Press: Cambridge, MA, USA.

[7] Keith S. Decker and Victor R. Lesser. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346, 1992.

[8] A. Drogoul. When ants play chess (or can strategies emerge from tactical behviours?). In C. Castelfranchi and J.-P. Müller, editors, *From Reaction*

to Cognition — Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-93 (LNAI Volume 957), pages 13–27. Springer-Verlag: Heidelberg, Germany, 1995.

[9] Edmund H. Durfee. Organizations, plans, and schedules: An interdisciplinary perspective on coordinating ai agents.

[10] David E. Foulser, Ming Li, and Qiang Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2–3):143–181, 1992.

[11] Shaw Green, Leon Hurst, Brenda Nangle, Pádraig Cunningham, Fergal Somers, and Richard Evans. Software agents: A review. Technical Report TCS-CS-1997-06, Dublin, 1997.

[12] Haddawy and S. Hanks. Utility models for goal-directed decision theoretic planners. Technical Report TR-93-06-04, 1993.

[13] Patrik Haslum and Peter Jonsson. Some results on the complexity of planning with incomplete information. In *ECP*, pages 308–318, 1999.

[14] N. R. Jennings. Coordination techniques for distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley & Sons, 1996.

[15] El Fallah-Seghrouchni Lip. Modelling, control and validation of multi-agent plans in dynamic context.

[16] J. Scott Penberthy and Daniel S. Weld. Temporal planning with continuous change. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1010–1015, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[17] M. Peot and D. Smith. Conditional nonlinear planning. In James Hendler, editor, *Proceedings of the First International Conference on AI Planning Systems*, pages 189–197, College Park, Maryland, June 15–17 1992. Morgan Kaufmann.

[18] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[19] David E. Smith and Daniel S. Weld. Temporal planning with mutual exclusion reasoning. In *IJCAI*, pages 326–337, 1999.

[20] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[21] Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[22] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

Ich erkläre hiermit gemäss 17 Abs. 2 APO, dass ich die vorstehende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

(Datum) (Unterschrift)