



Karlsruher Institut für Technologie

ZUSAMMENFASSUNG
Lernende und Planende Roboter
(Robotik II)

Sophie v. SCHMETTOW
SoSe 15

Inhaltsverzeichnis

1 Einführung und Grundlagen	4
1.1 Klassifizierung der Verfahren zur Roboterprogrammierung	4
1.2 Arten der Programmierung	4
1.2.1 Direkte/Prozessnahe Programmierung	4
1.2.2 Textuelle Verfahren	7
1.2.3 Graphische/Gemischte Verfahren	7
1.3 Abstraktionsgrad der Programmierung	8
1.3.1 Explizite/Roboterorientierte Programmierung	8
1.3.2 Implizite/Aufgabenorientierte Programmierung	11
2 Aufgabenorientierte Programmierung	11
2.1 Umweltmodellierung	11
2.1.1 Objektmodell	11
2.2 Aufgabenmodellierung	17
3 Interaktive Programmierung: Programmierung durch Vormachen von Manipulationsaufgaben	23
3.1 Grundlagen	23
3.2 PdV - Generelles Framework	24
3.3 Klassifikation von PdV-Verfahren	25
3.4 Prozesskomponenten der interaktiven Programmierung	27
3.5 Wissensrepräsentation: Beispielsysteme	29
3.6 Bahnplanung	36
3.7 Griffklassifikation	44
3.8 Griffplanung	45
4 Aktionsplanungsverfahren	48
4.1 Definitionen	48
4.1.1 Planungsproblem	48
4.1.2 Aktionsplanung	48
4.2 Suchverfahren	49
4.2.1 Uninformierte Suche: Analytische Verfahren (Baumsuche)	50
4.2.2 Informierte Suche: Heuristiken	50
4.3 Lineare Planung	51
4.3.1 Blockwelt	51
4.3.2 Formalisierung: Situationskalkül	52
4.3.3 STRIPS	53
4.3.4 Diskussion	55
4.4 Nichtlineare Planung	55
4.4.1 Einfacher Algorithmus	56
4.4.2 Partial Order Planning (POP)	56
4.4.3 Total vs. Partial Order Planning	58
4.5 Hierarchische Planung	58
4.5.1 HTNs – Suchraum	59
4.5.2 Simple Hierarchical Order Planner (SHOP)	59
5 Probabilistisches Entscheiden	60
5.1 Der rationale Agent	61
5.1.1 Beispiele	61
5.1.2 Umwelt	62
5.1.3 Utility – Motivation des Agenten	63
5.1.4 Problem	64

Inhaltsverzeichnis

5.2	Markov Prozesse	64
5.2.1	Markov Entscheidungsprozesse	65
5.2.2	MDP Value Iteration	69
5.2.3	MDP – Schlussfolgerung	72
5.3	POMDP (Partially Observable Markov Decision Processes)	72
5.3.1	Eigenschaften und Ablauf von POMDPs	75
5.3.2	POMDP Value Iteration	78
5.3.3	POMDPs und Serviceroboter	80

Disclaimer

Dieses Dokument wurde im Rahmen des Master-Studiums für das Modul „Autonome Robotik“ erstellt. Es stellt eine Zusammenfassung dar und dient zur Vorbereitung auf die mündliche Prüfung. Neben den Materialien der Vorlesung fließen auch weitere Quellen ein, um den behandelten Stoff auszuarbeiten. Auf den Verweis von Quellen wird verzichtet, da die Erstellung keinerlei wissenschaftlichen Zweck verfolgt und nur für den privaten Gebrauch bestimmt ist.

1 Einführung und Grundlagen

1.1 Klassifizierung der Verfahren zur Roboterprogrammierung

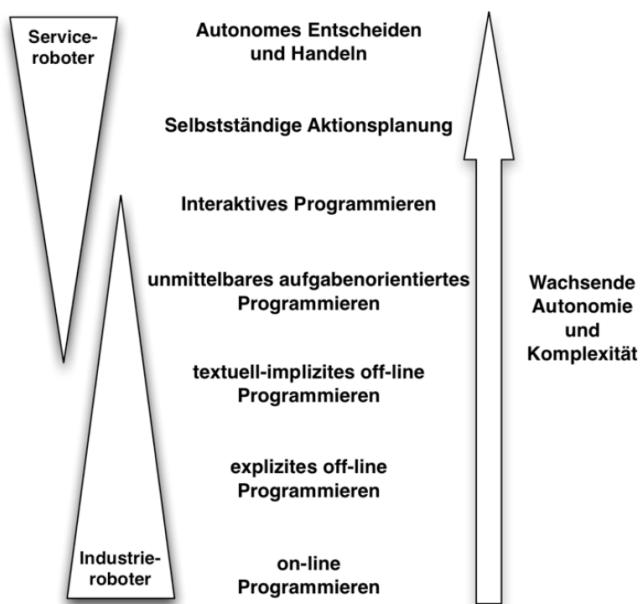


Abbildung 1: Überblick

Kriterien:

- Programmierort:** On-line (prozessnah, am Roboter) vs. Off-line (prozessfern, ohne Roboter)
- Art der Programmierung:** Direkte vs. indirekte (textuelle oder graphische/gemischte) Programmierung, hybride Verfahren
- Abstraktionsgrad der Programmierung:** Explizite/bewegungs- bzw. roboterorientierte vs. implizite/aufgabenorientierte Programmierung

1.2 Arten der Programmierung

1.2.1 Direkte/Prozessnahe Programmierung

- **Einstellen des Roboters:** Ältestes Programmierverfahren. Der Bewegungsbereich jedes Gelenks wird durch Stopper eingeschränkt. Die Bewegungen erfolgen für jedes Gelenk einzeln bis zum Anschlag. Zuordnung zwischen Anfahrpunkten zu Stopper kann mit Codiermatrizen erfolgen. (sogenannter „Bang-Bang-Robot“)
Nachteil: sehr kleine Menge von Anfahrpunkten.
- **Teach-In Programmierung:** Anfahren markanter Punkte der Bahn mit manueller Steuerung (Teach Box, Teach Panel, weitere: Spacemouse, Teach-Kugel)
Funktionalität einer Teach Box:

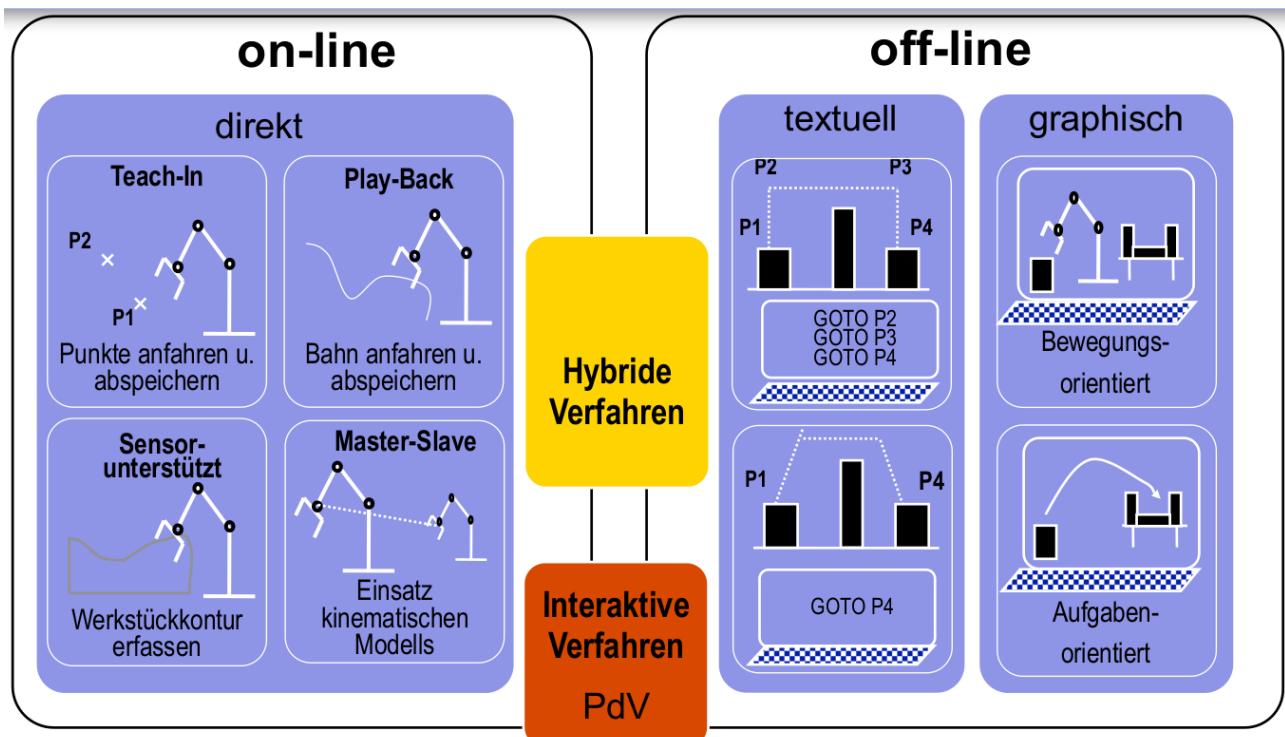


Abbildung 2: Schema

- Einzelbewegung der Gelenke
- Bewegung des Effektors in 6 Freiheitsgraden
- Punkte anfahren u. abspeichern
- Speichern / Löschen von Anfahrpunkten
- Eingabe von Geschwindigkeiten
- Eingabe von Befehlen zur Bedienung des Greifers
- Starten / Stoppen ganzer Programme

Vorgehensweise beim Teach-In:

- Anfahren markanter Punkte der Bahn (= Folge von Zwischenpunkten)
- Speichern der Gelenkwerte
- Ergänzung der gespeicherten Werte um Parameter wie Geschwindigkeit, Beschleunigung usw.
- Anwendung: in der Fertigungsindustrie (Punktschweißen, Nieten), Handhabungsaufgaben (Pakete vom Fließband nehmen)

• Play-Back- (manuelle) Programmierung:

- Einstellung des Roboters auf Zero-Force-Control (Roboter kann durch den Bediener bewegt werden)
- Abfahren der gewünschten Bahn
- Speichern der Gelenkwerte: automatisch (definierte Abtastfrequenz) oder manuell (durch Tastendruck)
- Anwendung: mathematisch schwer beschreibbare Bewegungsabläufe, Integrierung der handwerklichen Erfahrung, typischerweise für Lackieren oder Kleben eingesetzt

Nachteile
<ul style="list-style-type: none"> - schwere Roboter schwierig zu bewegen - wenig Platz in engen Fertigungszellen für Bediener, dadurch Sicherheitsrisiko - hoher Speicherbedarf (bei hoher Abtastrate) - schlechte Korrekturmöglichkeiten

Tabelle 1: Nachteile der Play-Back-Programmierung

- **Master-Slave-Programmierung:**

- Bediener führt einen kleinen, leicht bewegbaren Master Roboter (entspricht einem kinematischen Modell des Slave-Roboters)
- Bewegung wird auf den Slave-Roboter übertragen
- Bewegungen werden synchron ausgeführt
- Slave-Roboter wirkt als Kraftverstärker
- Anwendung: Handhabung großer Lasten bzw. großer Roboter

Vorteile	Nachteile
<ul style="list-style-type: none"> + Möglichkeit, auch schwerste Roboter zu programmieren 	<ul style="list-style-type: none"> - teuer, da zwei Roboter benötigt werden

Tabelle 2: Zusammenfassung Master-Slave-Programmierung

- **Sensorunterstützte Programmierung:**

Manuell

- Bediener führt Programmiergriffel (Leuchtstift, Laserstift) entlang der abzufahrenden Bahn
- Erfassung der Bewegung durch externe Sensoren (zB. Kameras, Laserscanner)
- Berechnung der inversen Kinematik
- Abspeichern der Bahn als Folge der Gelenkwinkel

Automatisch

- Vorgabe des Start- und Zielpunktes
- Sensorische Erkennung der Sollkontur (zB. über Kraft-Momenten-Sensor)

Anwendung: Schleifen, Entgraten von Werkstücken

Vorteile	Nachteile
<ul style="list-style-type: none"> + schnell bei einfachen Trajektorien + sofort anwendbar + geringe Fehleranfälligkeit + Bediener benötigt keine Programmierkenntnisse + kein Modell der Umwelt erforderlich 	<ul style="list-style-type: none"> - hoher Aufwand bei komplexen Trajektorien - nur mit und am Roboter möglich - spezifisch für einen Robotertyp - Verletzungsgefahr durch Roboter

Tabelle 3: Zusammenfassung Direkte Programmierung

1.2.2 Textuelle Verfahren

Erstellung von Robotersteuerprogrammen erfolgt mittels erweiterter, höherer Programmiersprachen (PasRo, Val, etc.)

Sprache: (DIN 66025)

- Programm = Menge nummerierter Sätze
z.B. „N70 G00 X20 Z12“ entspricht Werkzeug im Eilgang (G00) an Position X=20 Z=12 bewegen.
(N = Satznummer)
- Sprachen:
 - APT (Automatically Programmed Tools), 1961 MIT
 - EXAPT (Extended Subset of APT), 1966 Aachen

Vorteile	Nachteile
<ul style="list-style-type: none"> + Programmierung kann unabhängig vom Roboter erfolgen + strukturierte, übersichtliche Programmierlogik + Erstellung komplexer Programme (Einbezug von Wissensbasis, Weltmodell, Auswertung von Sensoren) 	<ul style="list-style-type: none"> - Bediener benötigt Programmierkenntnisse - keine / schlechte Korrekturmöglichkeiten

Tabelle 4: Zusammenfassung Textuelle Verfahren

1.2.3 Graphische/Gemischte Verfahren

- Graphische Darstellung von Kontrollstrukturen (if/else, Schleifen, Marken...)
- Kopplung mit Visualisierungs-Tool und Simulations-Tool (z.B. RobCAD)
- Trajektorien durch Interpolation aus Stützpunkten, Freihandzeichnen, analytisch
- Operationen Icon- oder Menu-gesteuert
- Simple graphische Programmierung, z.B. Lego Mindstorms: Leicht verständliche, ikonische Programmierung aber beschränkte Möglichkeiten
- Graphische Programmierung basierend auf sensorieller Erfassung der Benutzervorführung: Simulation der Roboterprogramme

Vorteile	Nachteile
<ul style="list-style-type: none"> + Programmierer benötigt weniger Programmierkenntnisse + einfache Programmierung, leichte Fehlererkennung + schnelles Erstellen komplexer Programme (rapid prototyping) 	<ul style="list-style-type: none"> - sensorielle Benutzererfassung noch zu ungenau - Leistungsfähige Hardware für Signalanalyse, Modellierung, ... - Komplexe Modelle benötigt - 2D-Sicht des Anwenders

Tabelle 5: Zusammenfassung Graphische Verfahren

1.3 Abstraktionsgrad der Programmierung

1.3.1 Explizite/Roboterorientierte Programmierung

„Wie ist es zu tun?“

Bewegungen und Greiferbefehle sind direkt in eine Programmiersprache eingebunden.

Das Aufgabenmodell ist gegeben durch Anfangs- und Endzustand (z.B. Relationale Darstellung). Ein Beispiel ist in Abbildung 3 dargestellt.

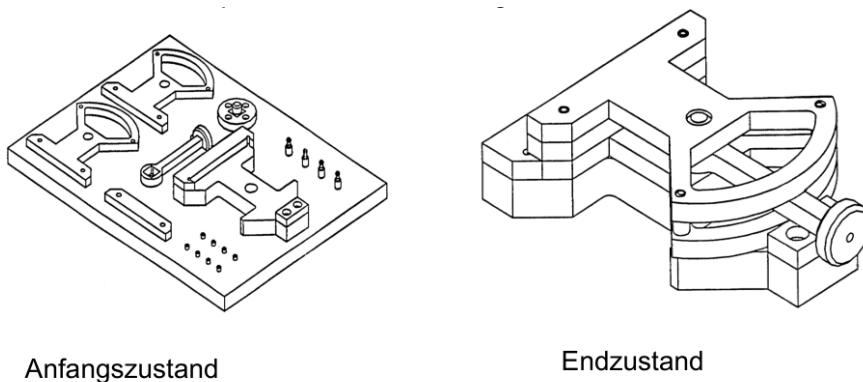


Abbildung 3: Der Cranfield-Montage-Benchmark

Anwender → Roboter /vgl. Abbildung 4 Jedes Robotersystem besitzt eine roboterabhängige Steuerungsebene, welche folgende Eigenschaften kapselt:

- Ansteuerung der Hardware (sowohl interne als auch externe)
- Bewegungsaktionen
- Lokale Modelle
- Elementare Operationen (erfordern evtl. Echtzeitregelung)

Anforderungen an die roboterorientierte Programmierung

- Positions-, Geschwindigkeitsregelung der aktiven Komponenten (z.B. Gelenke, Räder)
- Auslesen und Parametrieren der internen und externen Sensorik
- Koordinatentransformationen, direkte und inverse Kinematik

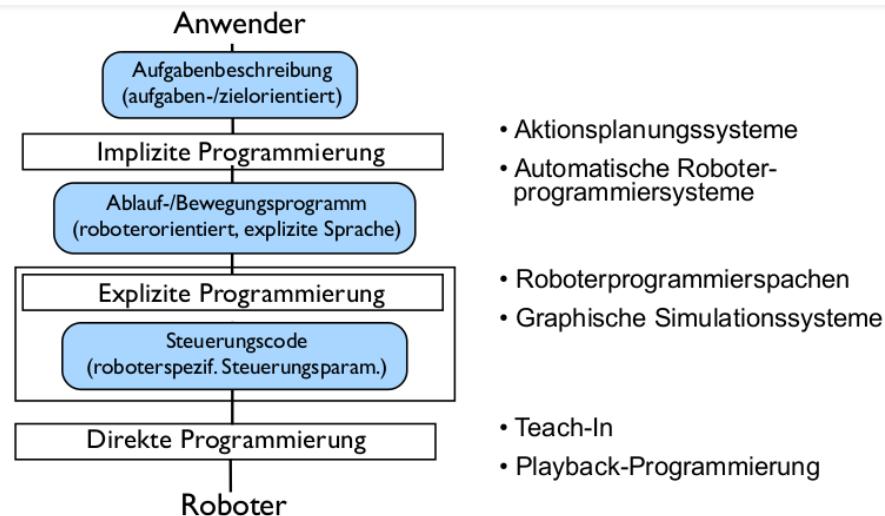


Abbildung 4: Einordnung roboterorientierte Programmierung

- Sensorabhängige Regelung
- Verfahren auf Trajektorien
- Generierung und Zusammensetzen von Trajektorien zu komplexen Bewegungen
- Verkettung komplexer Bewegungen zu Elementaroperationen

Komponenten der roboterorientierten Programmierung Abbildung 5

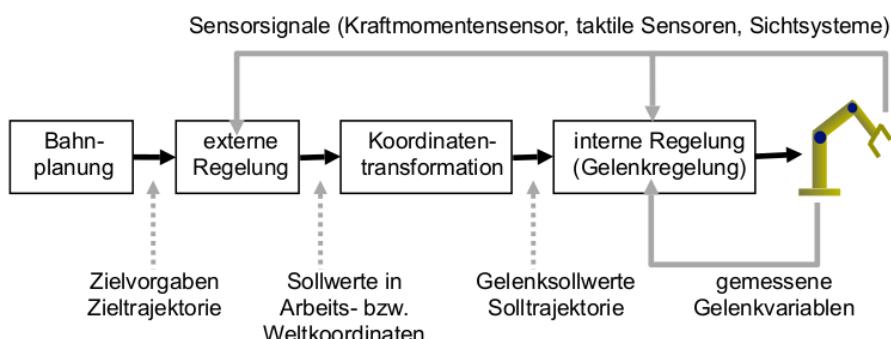


Abbildung 5: Regelungszyklus eines Roboters

Sprachelemente von Roboterprogrammiersprachen

- Befehle für:
 - Bewegung eines oder mehrerer Roboter
 - Betrieb von Greifern / Werkzeugen
 - Ein-/Ausgabe von Daten / Signalen über Schnittstellen
 - Externe Sensoren
 - Zur Synchronisation / Kommunikation zwischen Prozessen
 - Parallelverarbeitung
 - Zur logischen Verkettung von Koordinatensystemen

- Anweisungen zur Ablaufsteuerung
- Definition generischer Operationen (z.B. Armbewegung + Griff → ein Operator)

Bewegungsanweisungen

- Bewegungen im Gelenkwinkelraum:
 - Bewege alle Gelenke mit max. Geschwindigkeit
 - Regelung der Geschwindigkeiten mit gleichzeitiger Beendigung der Bewegung aller Gelenke
- Kartesische Bewegungen (Stellung des TCPs)
 - Erfordert inverse Kinematik
 - Nutzung von Frames, z.B. relativ zu einem Objekt
- Geometriebezogene Bahndefinition

Vorteile	Nachteile
<ul style="list-style-type: none"> + Hohe Einstellgenauigkeit der Position + Hohe Wiederholgenauigkeit + eindeutige Roboterkonfiguration + keine inverse Kinematik erforderlich 	<ul style="list-style-type: none"> - Abhängigkeit von Robotertyp - kein Bezug der Gelenkwinkel zur Objektlage

Tabelle 6: Bewegungen im Gelenkwinkelraum

Sprachelemente – Semantik von Greiferbefehlen

- Verschiedene Greifertypen: evtl. mit taktiler und/oder Kraftsensorik
- Backengreifer: Industrie, Forschung (z.B. 3-Finger-Hand)

Mit zunehmender Abstraktion:

1. Steuerung im Gelenkwinkel-Raum
 - Anzahl der Freiheitsgrade bestimmt Anzahl der Parameter (Backengreifer: 1, menschl. Hand: 22)
 - Wenig Kapselungs-Aufwand (keine inverse Kinematik nötig)
 - Hand-abhängig
 - Kein Bezug zwischen Fingerstellung und Handstellung
2. Steuerung im kartesischen/ zylindrischen/... Raum
 - Parameter: Position/Orientierung jedes Fingers/ jeder Fingerspitze
 - Inverse Kinematik nötig
 - Berechnung aus Greif-/ Bewegungsplanung oder aus menschlicher Vorführung
 - Mögliche Konfigurationen (Konfigurationsraum) handabhängig
3. Semantische Steuerung
 - Parameter: Griff-Form, zu greifendes Objekt, Objektgröße, Objektform, ...
 - Mapping auf Roboterhand nötig
 - Übertragbar auf andere Roboterhände (mögliche Griffe handabhängig)
 - Griff-Form vom Menschen lernbar

Zusammenfassung – Explizite Programmierung Nur in Verbindung mit (abstrakten) Programmiersprachen

Vorteile	Nachteile
<ul style="list-style-type: none"> + beliebig komplexe Bahnen + Anbindung von Sensoren + reaktive Planung 	<ul style="list-style-type: none"> - Keine standardisierte Programmiersprache - Kenntnis der Programmiersprache

Tabelle 7: Bewegungen im Gelenkwinkelraum

1.3.2 Implizite/Aufgabenorientierte Programmierung

„Was ist zu tun?“

Die Aufgabe, die der Roboter durchführen soll, wird beschrieben, z.B. in Form von Zuständen.

- Abstrakte Form der Programmierung erfolgt in den Phasen
 1. Modellierung der Umwelt
 2. Spezifikation der Aufgaben
 3. Erzeugung der Roboterprogramme
- u. U. erfolgt vor der Ausführung eine Überprüfung des Roboterprogramms (Simulation)
- Beispiel: Einschenken unter Berücksichtigung von Hindernissen
- Details im kommenden Kapitel

2 Aufgabenorientierte Programmierung

2.1 Umweltmodellierung

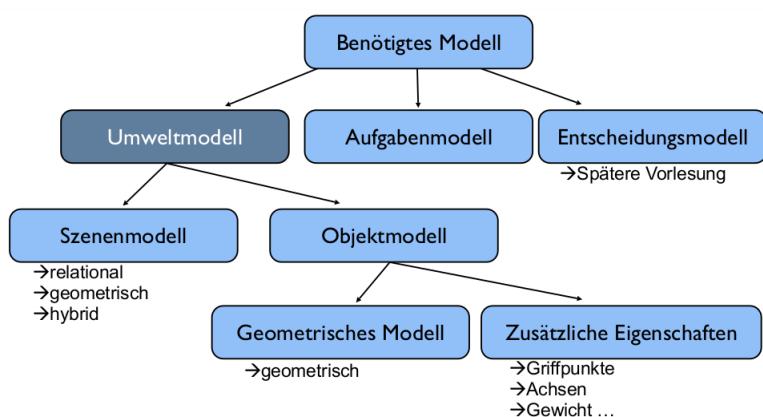


Abbildung 6: Umweltmodell

2.1.1 Objektmodell

Die geometrische Beschreibung von Objekten beinhaltet:

- graphische Darstellung
- Kollisionsberechnungen, Kontaktberechnung in Griffplanung, ...
- physikalisch-dynamische Simulation der Effekte von Handlungen auf die Umwelt

- geometriebezogene Bewegungsplanung

Es gibt drei Ansätze zur geometrischen Modellierung (Abbildung 7):

1. Kantenmodelle
2. Flächenmodelle
3. Volumenmodelle

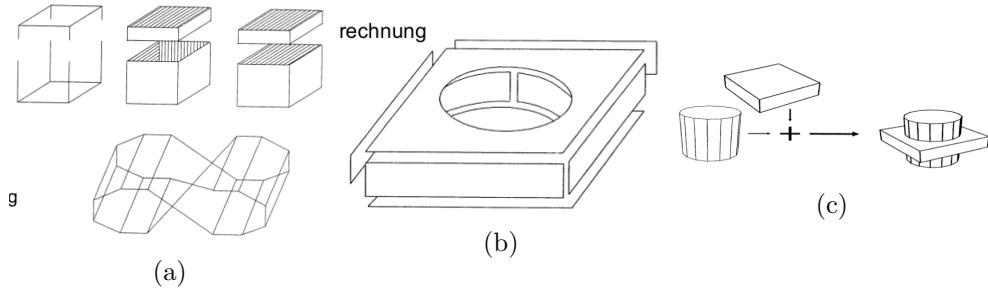


Abbildung 7: Objektrepräsentation

Kanten Nur die Kanten werden gespeichert, d.h. Punkte und Verbindungen (Gerade, Polygonzug, Bezierkurve, ...).

Vorteile	Nachteile
<ul style="list-style-type: none"> + einfache Daten + wenige Daten 	<ul style="list-style-type: none"> - Mehrdeutigkeiten (Abb. 7a unten) - hoher Eingabearaufwand - keine Kollisionsberechnung - kein Schnitt

Tabelle 8: Zusammenfassung – Kantenmodelle

Flächen Flächen können exakt modelliert werden, wenn sie **analytisch gegeben** sind (eine 3D Kugel beispielsweise durch $r = \|x - p\|$, wobei r der Radius und p der Mittelpunkt ist).

Vorteile	Nachteile
<ul style="list-style-type: none"> + Geschlossene Darstellung (wenig Speicherbedarf) + Analytische Darstellung erlaubt einfache Rechenverfahren (z.B. Schnitt von Ebenen / Kugeln → schnelle Kollisionsberechnung) 	<ul style="list-style-type: none"> - Wenige Flächen sind analytisch darstellbar

Tabelle 9: Flächen – analytisch

Ansonsten werden sie **approximativ** durch Bildung einer großen Fläche aus einem Netz („Mesh“) von einfachen Einzelflächen (z.B. Dreiecke, Vierecke) modelliert.

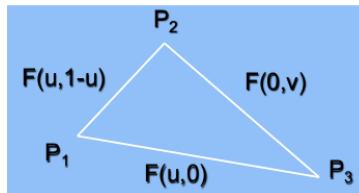
Vorteile	Nachteile
+ Definition sehr einfach + einfache Algorithmen	- hoher Speicherbedarf - hoher Rechenaufwand

Tabelle 10: Flächen – approximativ

Hierbei werden Freiformflächen im einfachsten Fall durch **Dreiecksflächen** approximiert:

Gegeben seien 3 Punkte im Raum P_1, P_2, P_3 .

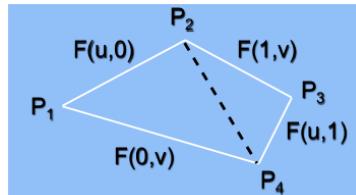
Damit hat die Fläche folgende Gleichung: $F(u, v) = u \cdot P_1 + v \cdot P_2 + (1 - u - v) \cdot P_3$ mit $0 \leq u, v, u + v \leq 1$.



Oder sie werden durch **Bilineare Viereckselemente / Pflaster** approximiert:

Gegeben sind 4 Punkte im Raum P_1, P_2, P_3, P_4

Damit wird die Fläche definiert durch $F(u, v) = (1-u)(1-v) \cdot P_1 + (1-u)v \cdot P_2 + u(1-v) \cdot P_3 + uv \cdot P_4$ mit $0 \leq u \leq 1, 0 \leq v \leq 1$.



Vorteile	Nachteile
+ Flächenelemente können gekrümmmt sein → weniger Gitterpunkte bei gleich guter Approximation	- Rechnen mit gekrümmten Flächen ist aufwendig

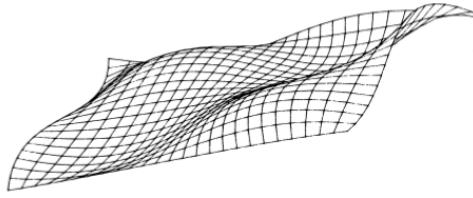
Tabelle 11: Approximation durch Vierecke

Zudem können Flächen durch **Beziersflächen**, einer Erweiterung der Bezierkurven beschrieben werden:

Gegeben ist ein Gitter von Führungspunkten $P_{ij}, 0 \leq i \leq N$ und $0 \leq j \leq M$.

Damit ist die Fläche beschrieben durch $F(u, v) = \sum_{i=0}^N \sum_{j=0}^M P_{ij} \cdot B_{i,N}(u) \cdot B_{j,M}(v)$ mit $B_{i,N}(u) = (1-u)B_{i,N-1}(u) + uB_{i-1,N-1}(u)$ und $B_{j,M}(v) = (1-v)B_{j,M-1}(v) + vB_{j-1,M-1}(v)$.

Die $B_{i,N}$ bzw. $B_{j,M}$ heißen auch Bernsteinpolynome.



Vorteile	Nachteile
<ul style="list-style-type: none"> + effiziente Verfahren + entspricht dem Vorgehen während der Modellierung + schnelle Kollisions- und Abstandsberechnung 	<ul style="list-style-type: none"> - hoher Eingabearaufwand - Darstellung aufwendig - Problem bei Schnittoperationen - Inkonsistenzen möglich

Tabelle 12: Zusammenfassung – Flächenmodelle

Volumen Vier verschiedene Arten von Volumenmodellen:

Parametrische Modelle:

Grundkörper und topologische Operationen auf diesen (Schnitt, Vereinigung, ...) werden abgespeichert.

Vorteile	Nachteile
<ul style="list-style-type: none"> + eindeutige Objektbeschreibung + geringer Eingabearaufwand + Ergebnis von Operationen sind korrekte Objekte 	<ul style="list-style-type: none"> - hoher Implementierungsaufwand - Einbindung von Freiformflächen schwierig

Tabelle 13: Volumenmodelle – parametrisch

Die Objekte sind bereits vorhanden und können durch Angabe von Parametern angepaßt werden (Varianten).

Konsistenzprüfungen sind notwendig (Abbildung 8)!

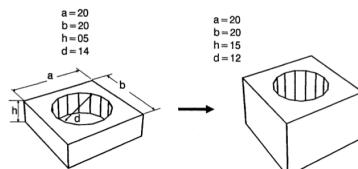


Abbildung 8: Konsistenzprüfung: $d < \min(a, b)$

Zellenzerlegung:

Objekte werden aus disjunkten Elementarzellen aufgebaut. Verwendung finden einfache geometrische Objekte z.B. Tetraeder, Quader, ... Benutzt in der Strukturanalyse mit Finite-Elemente-Methoden (FEM).

Diese Modelle können auf drei Arten umgesetzt werden:

1. Boundary Repräsentation
2. Constructive Solid Geometry (CSG)
3. Zellenbelegung

Boundary Repräsentation: Hierarchische Darstellung eines Objektes durch begrenzende Elemente, i.d.R. Kanten oder Flächen. Abbildung 9 zeigt ein Beispiel.

Vorteile: aus der topologischen Struktur Information über z.B.:

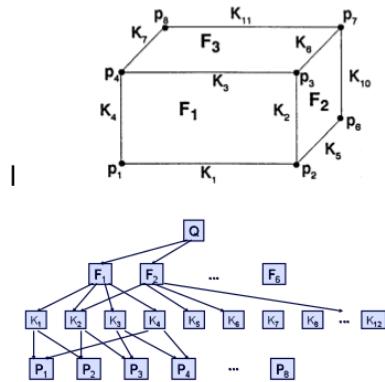


Abbildung 9: Elemente eines Quaders im Flächenmodell: Quader Q , Flächen $F_i : i \in \{1, \dots, 6\}$, Kanten $K_i : i \in \{1, \dots, 12\}$, Ecken $P_i : i \in \{1, \dots, 8\}$

- Welche Flächen gehören zum Objekt?
- Welche Kanten gehören zur Fläche? → kantenbasierte Objekterkennung
- Zu welchem Objekt gehört eine Fläche?
- Zu welchem Objekt gehört eine Kante?
- Welche Flächen stoßen aneinander?

Constructive Solid Geometry (CSG): Es gibt eine Menge von einfachen Grundkörpern, die parametriert werden können (Abbildung 10). Auf ihnen sind verschiedene Operationen definiert, z.B.

	Objekte A & B
Vereinigung $A \cup B$ (Summe)	
Schnitt $A \cap B$	
Differenz A/B	
Sweep: Ein Grundelement (u.U. eine Fläche) wird entlang einer Raumkurve verschoben. Der durchdringene Raum stellt das neue Objekt dar.	

Tabelle 14: CSG – Operatoren

Grundkörper	Parameter	Skizze
Quader	Länge, Breite, Höhe	
Prisma	Länge, Radius, Facetten	
Zylinder	Länge, Radius	
Kegel	Länge, Radius a, Radius b	
Ellipsoid	Radius a, Radius b, Radius c	
Rotationskörper	Achse, Kontur	

Abbildung 10: Constructive Solid Geometry

Zellenbelegung: Der Raum wird in mehrere Zellen unterteilt (i.d.R. 8 Zellen: „Octree“). Wenn eine Zelle komplett vom Objekt belegt ist, als „belegt“ markieren. Wenn die Zelle nur teilweise belegt ist, dann wird auf diese Zelle das Verfahren rekursiv angewendet. Ansonsten ist die Zelle leer. Die Rekursion terminiert bei einer vorbestimmten minimalen Zellgröße. Teilbelegte kleinste Zellen werden als belegt markiert. Siehe Abbildung 11.

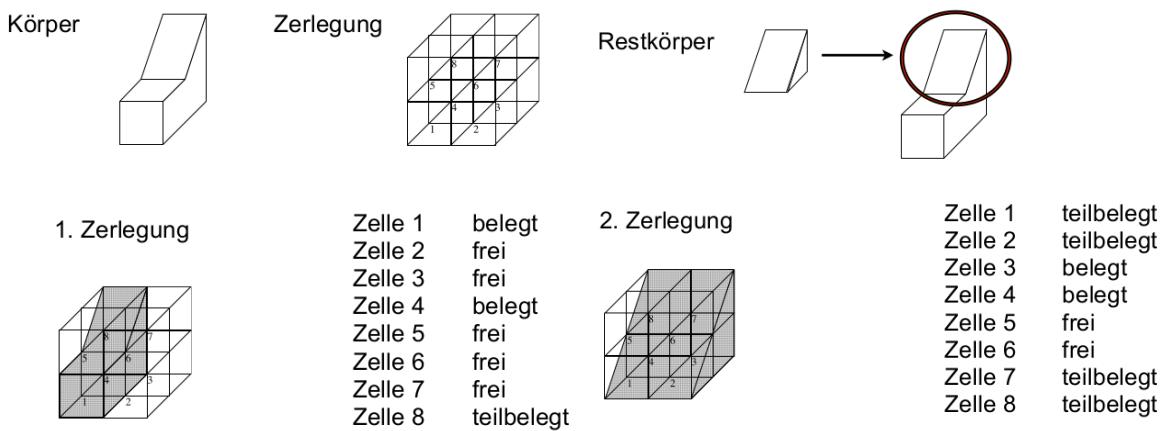


Abbildung 11: Beispiel – Zellenbelegung

2.2 Aufgabenmodellierung

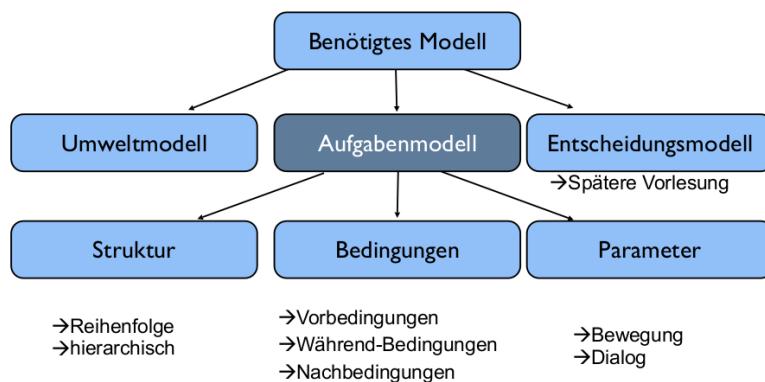


Abbildung 12: Aufgabenmodell

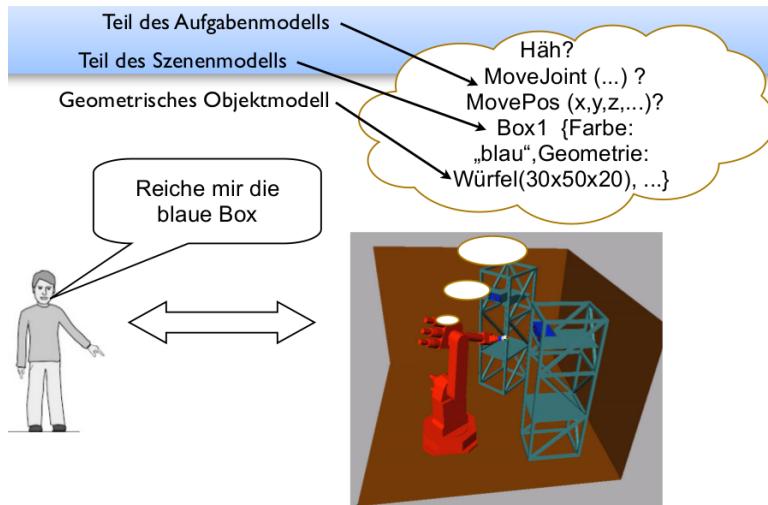


Abbildung 13: Beispiel

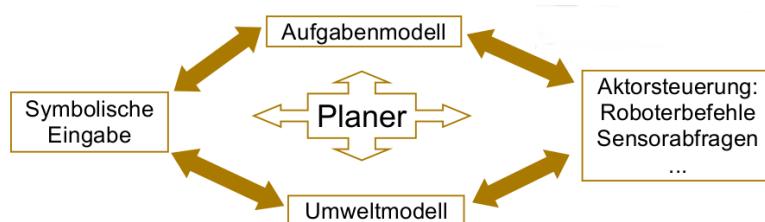


Abbildung 14: Einordnung des Aufgabenmodells

Anforderungen an das Aufgabenmodell:

- Erweiterbarkeit
- Erklärbarkeit
- Wiederverwendbarkeit
- Integration des Wissens in ein Planungssystem

Symbolische Abstraktion Benutzer beschreibt die Aufgabe mit seinen Worten (z.B. „Bring mir Tee!“)

- Abbildung: Erzeugung der Roboterbefehle über mehrere Stufen (vgl. Abbildung 15)
- „Fahre in die ‚Küche‘, greife ein Glas, fahre zurück und reiche mir den Becher.“
- „DriveTo(...), SearchObject(...), MoveArm(...), Grasp(Becher), MoveArm(...), DriveTo(...), MoveArm(...) ...!“

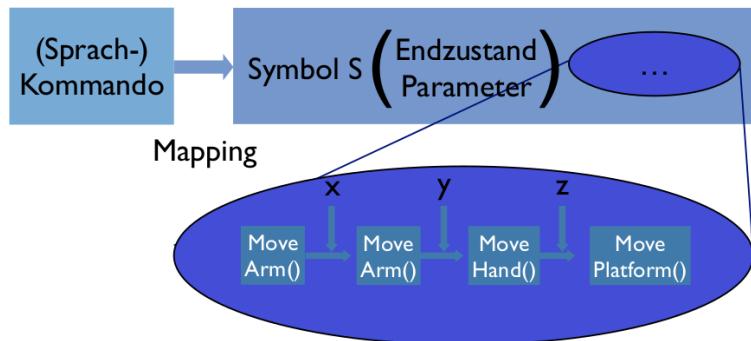


Abbildung 15: Symbolische Abstraktion

Modellierung der Reihenfolge von Operatoren/Symbolische Handlungsatome

- In den gegebenen Beispielen wurden bereits implizit Handlungsatome angenommen
- Die kleinsten auszuführenden Handlungseinheiten werden als Elementaroperationen oder atomare Handlungen bezeichnet.
- Komplexe Handlungen werden aus Elementaroperationen zusammengesetzt. Diese können dazu üblicherweise parametriert werden.
- Jeder Roboter besitzt eine endliche Menge an Elementaroperationen.

Drei Ansätze für das Aufgabenmodell

1. Sequentiell (Abbildung 16a): Festgelegte Folge von elementaren Aktionen
2. Vorranggraph (Abbildung 16b): Darstellung der Abhängigkeiten
3. Hierarchisch (Abbildung 16c): Abstraktion von Teilhandlungen

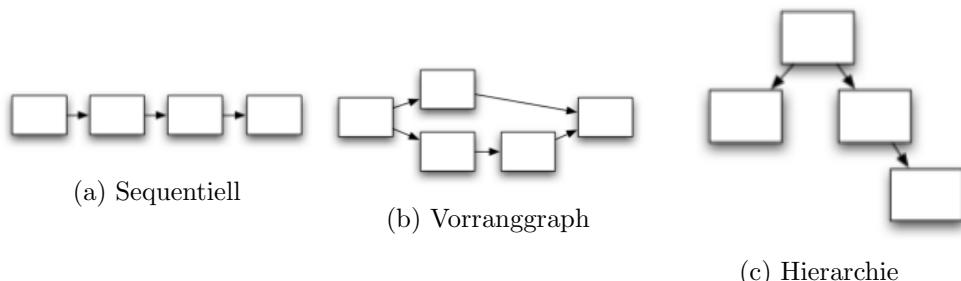
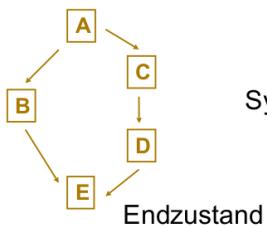


Abbildung 16: Ansätze für das Aufgabenmodell

Sequentielle Handlungsbeschreibung:

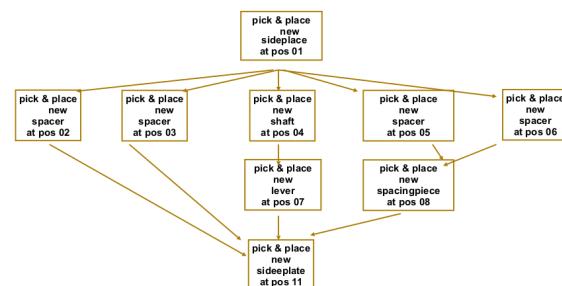
- Folge von (parametrierten) atomaren Handlungen
- Eindeutig festgelegte Reihenfolge
- Reihenfolge und Anordnung nicht unbedingt erklärbar (lesbar)
- Bedingungen, Alternativen, etc. schlecht darstellbar
- Sinnvoll für einfache Aufgaben oder sehr strukturierte Umgebungen (z.B. Leittechnik/Industrie)
- Komplexe Handlungen: Rein sequentielle Beschreibungen nicht mächtig genug!
- Ausführung sequentiell beschriebener Handlungen trivial
- Linearer Handlungsfluss
- Ausführung besteht aus: Anstoßen einer Elementarhandlung, evtl. Handlungsüberwachung und nach (erfolgreicher) Beendigung Übergang zur nächsten Elementarhandlung
- Einfache Mechanismen zur Ausführung nötig!

Vorranggraph:



Symbol S **(Endzustand Parameter)**

vor „B“ muß „A“ ausgeführt sein
 vor „C“ muß „A“ ausgeführt sein
 vor „E“ müssen „B“ und „D“ ausgeführt sein
 keine Aussage zwischen „B“ und „C“ bzw. „B“ und „D“
 (a) Modellierung der Reihenfolge: Vorranggraph



(b) Beispiel: Vorranggraph des Cranfield-Benchmarks (simple Montage-Aufgabe)

Abbildung 17: Vorranggraph

- Darstellung der Handlung in mehreren unabhängigen Teilzweigen
- Üblicherweise: Serialisierung notwendig!
 - Berechnung optimaler Operatorreihenfolgen
 - Freiheitsgrad zum Ausführungszeitpunkt
 - Serialisierung aufwendig (→ Planungsverfahren, siehe spätere Vorlesungen)
- Ausführung beinhaltet also:
 - Serialisierung der Handlung nach gegebenen Kriterien
 - Ausführung der sequentiell beschriebenen Handlung
- Mächtigere Handlungsbeschreibung benötigt auch mächtigere Verfahren bei der Ausführung!

Hierarchisches Aufgabenmodell (Abbildungen 18 und 19):

Sowohl Aufgabenspezifikation als auch Aufgabenzerlegung/-teilung sind hierarchisch strukturiert. Die Abstraktion erfolgt nach Raum, Zeit, Objekten und Alternativen.

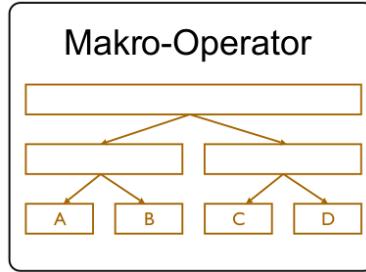


Abbildung 18

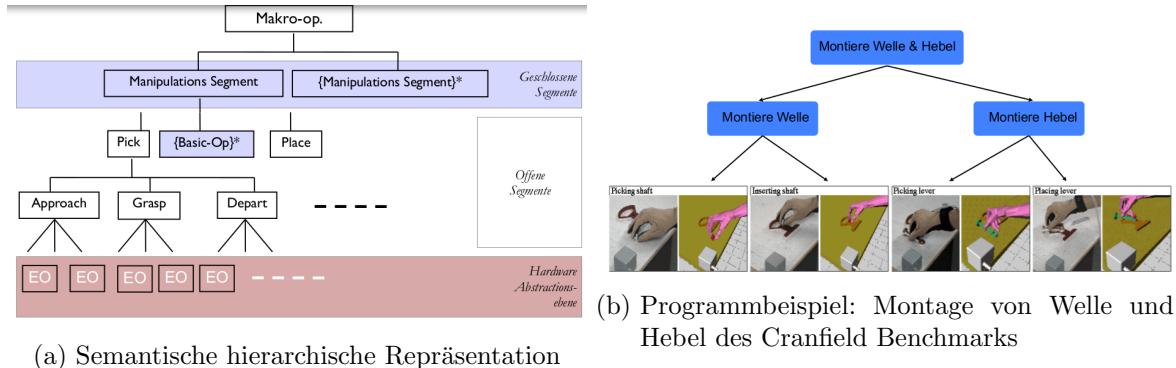


Abbildung 19: Hierarchisches Aufgabenmodell

Abstraktionsebenen im Aufgabenmodell Häufige Unterscheidung der in Abbildung 20 dargestellten semantischen Abstraktionsstufen. Diese benutzen symbolische Parameter.

- Komplexität der Aufgabe beherrschbar
- Formalismen anwendbar

Stufe	Ebene	Beispiel
1	Mission	<ul style="list-style-type: none"> • Exchange • Install
2	Task	<ul style="list-style-type: none"> • Pick Up • Place
3	Aktion	<ul style="list-style-type: none"> • Move • Grasp

Abbildung 20

Wahl der Abstraktionsebene: Wie wird die Abstraktionsebene der Aktion gewählt, damit Aktionen von Roboter A auf Roboter B übertragen werden können?

- Aktion: Kleinste symbolische Einheit
- Darunter: Regelungsebene
- Abhängig von:
 - Komplexität der Aufgabe
 - Grad der Kopplung einzelner Teilhandlungen
 - Hardware: Getrennt geregelte Komponenten werden üblicherweise auch mit getrennten Aktionen modelliert

- Planungssystem/Beobachtungssystem
- Grundsätzlich: Die Frage ist in der Forschung noch nicht eindeutig geklärt, immer noch ein Streitpunkt!

Modellierung von Aktionen / Tasks:

Ein **Operator** ist definiert durch $OP = (N, O, A, Par, K)$ mit

- N = Name des Operators (eindeutig)
- O = Liste der beteiligten Objekte
- A = Auswahlbedingung
- Par = Liste der Parameter, z.B. Positionen, Kräfte, Beschleunigungen ...
- K = Körper des Operators:
 - **Aktion/Elementaroperator:**
ausführbares Programm
⇒ Realisierung einfacher Fähigkeiten
 - **Task/Makro-Operator:**
besteht aus weiteren Operationen $K = K_1K_2...K_n$
⇒ Realisierung komplexer Aktionen

Hierarchische Repräsentation – Ausführung:

- Im Aufgabenmodell ist die Serialisierung implizit gegeben
 - Minimaler Aufwand zur Ausführung
 - Aber: Bei Parallelitäten von Teilhandlungen: Konflikte möglich!
- Ausführung beinhaltet also:
 - Überprüfung auf Konflikte
 - Gegebenenfalls Serialisierung/Lösen der Konflikte
 - Ausführung der resultierenden Operatorsequenz
- Vorteil: Zusammengesetzte (Teil-)Programme können wiederverwendet werden
- Repräsentation gut „lesbar“ (verständlich, erklärbar)

Anwendung hierarchischer Handlungsbeschreibung in der Realität

– Flexible Programme:

Symbolische Abstraktionsebene zur Roboterprogrammierung

- Hierarchische Handlungsbeschreibung
- Erklärbar, verständlich, intuitiv
- Wiederverwendbarkeit
- Parametrierbar
- Unterstützt: Bedingungen, Verzweigungen, Ressourcenverwaltung, Parallelität
- Aufbau flexibler Programme:
 - Repräsentation des Handlungswissens als Baumstruktur
 - Parametrierbare Aktionsbeschreibung
 - Blätter entsprechen Roboteraktionen

- Abarbeitung entsprechend einer Tiefensuche
- Instantiierung der Kinder zur Laufzeit (Expansion des Baums)
- Auswahl des geeignetsten Kandidaten beim expandieren
- Parallele Ausführung mehrerer Kinder möglich

Validierung der Modelle: Simulation oder Graphische Animation

1. **Simulation der Komponenten:** Validierung anhand gegebener Einschränkungen (Kollisionen, Erreichbarkeit, Optimalitätskriterien: Weg, Zeit, Energie, ...)
 - Für die Simulation von Effekten durch Manipulatoren wird die Physiksimulation verwendet: Masse, Reibung, Kräfte, Gelenke
2. **Graphische Animation:** Der Anwender überprüft visuell die erstellten Modelle
 - Wenn die Robotersimulationen ergeben, dass die Zielstellung angefahren werden kann, wird die Bewegung in einer Animation graphisch dargestellt.

Aufgabenmodell – Zusammenfassung und Diskussion Zusammenfassung:

- Aufgabenmodell basiert auf elementaren Operationen
- Oft dreischichtiger Ansatz: Aktion, Task, Mission
- Verknüpfung der elementaren Operationen zu komplexen Aufgaben:
 - Sequentiell
 - Vorranggraph
 - Hierarchisch
 - (Kontrollstrukturen)
- Problem: Validierung der Programme
 - Simulation
 - Animation und Validierung durch den Menschen

Diskussion:

- Mobile Plattform ohne Sensorik (z.B. Roomba)
 - Aufgabenmodell?
 - Umweltmodell?
- Mobile Plattform mit Differentialantrieb und Sensorik zur Lokalisation (z.B. Transportaufgaben)
 - Aufgabenmodell?
 - Umweltmodell?
- Serviceroboter mit mobiler Plattform, Manipulator, Mehrfingergreifer, komplexe Sensorik
 - Aufgabenmodell?
 - Umweltmodell?
- Wo kommt jeweils das Aufgabenwissen dieser Systeme her?

3 Interaktive Programmierung: Programmierung durch Vormachen von Manipulationsaufgaben

3.1 Grundlagen

Neue Anforderungen an Robotersysteme

- In der **Produktion**: Klein- & Kleinstserienfertigung, Unikatfertigung (z.B. Prototyp)
 - Produkte mit: vielen Ausstattungsvarianten und hoher Rekonfigurierbarkeit
 - Flexible Fertigung
- Im **Servicebereich**:
 - Handel: Kommissionierung und Palettierung von Waren, Bestücken von Regalen
 - Pflege: Unterstützung von Rehabilitationsmaßnahmen durch Roboter, Rollstuhl mit Manipulationshilfe
 - Handwerk: Handhabungen in Schreinereien und Schlossereien
- In der **humanoiden Servicerobotik**: Manipulation beliebiger Objekte, selbstständiges Lösen komplexer Aufgaben, Einsatz im menschlichen Umfeld
 - komplexe Umgebung und sehr viele Freiheitsgrade
 - Wie Handlungswissen erzeugen?

Grundidee der interaktiven Programmierung

1. Mensch ist Domänenexperte (Manipulation)
2. Explizite Demonstrationen der Manipulationsaufgabe
3. Sensorielle Erfassung der Demonstrationen
4. Erzeugung der internen Repräsentation des Roboterprogramms
5. Abbildung auf das Robotersystem
6. Ausführung

Anforderungen an interaktive Programmierung

1. Intuitive Interaktionsformen: einfache Bedienung des Systems
2. Transparenz der Prozesse im Programmiersystem: Umsetzung von Handlungen des Benutzers soll nachvollziehbar sein
3. Abgleich von Systemhypthesen mit der Benutzerintention: z.B. zur Korrektur falscher Systemhypthesen
4. Flexibilisierung und Optimierung von Programmen: erlernte Programme sollen in vielen Situationen anwendbar sein
5. Wiederverwendung von Teillösungen: Funktionsbausteine sollen in anderen Programmen wieder verwendbar sein

Randbedingungen

Notwendig zur Erzeugung von leistungsfähigen, automatischen aber auch sicheren und komfortablen Roboterprogrammen

1. Weitgehend automatisierte Programmgenerierung
2. Beschränkung der Benutzerinteraktion auf das Nötigste

3. Maximierung des Informationsgewinns und der Eindeutigkeit der Ergebnisse der Benutzerinteraktion für das System
 4. Möglichst benutzerfreundliche Mensch-Maschine Interaktion: verständlich, transparent und flexibel, möglichst ähnlich zur zwischenmenschlichen Kommunikation

3.2 PdV - Generelles Framework

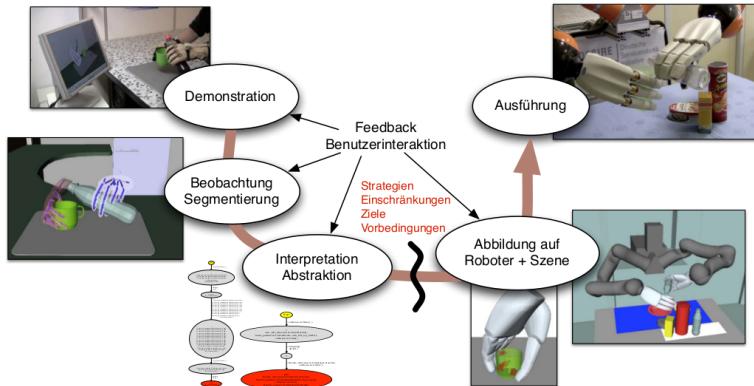


Abbildung 21: PdV-Zyklus

Abbildung 21 zeigt die Phasen von PdV.

Beobachtung

Beobachtung der Benutzerinteraktion mittels externer und interner Sensorik (z.B. Kamerasysteme, Datenhandschuhe, Positionstracker).

→ mit den gewonnenen Sensordaten lassen sich dann Objekte klassifizieren, Trajektorien aufzeichnen und weitere Vorverarbeitungsschritte wie Grifferkennung durchführen

Segmentierung

Segmentierung in relevante Operationen oder Umweltzustände.

- man benötigt hierfür die Trajektorien aus der Demonstration, eine Datenbasis mit den Sensordaten, einen Satz Aktionstypen (Griffe, Lageänderungen von Objekten), einen Satz von Elementaroperationen und das Umweltmodell
 - durch eine geeignete Heuristik lässt sich damit die Segmentierung durchführen
 - eine zusätzlich Interaktion mit dem Benutzer (über verschiedene Kommunikationskanäle wie grafische Benutzerschnittstellen oder Sprache) hilfreich; Rückfragen helfen bei der Identifikation der in der Lösung involvierten Objekte und sparen damit unnötige Berechnungen aller Relationen zwischen allen Objekten ein
 - Rauschfilterung

Interpretation / Abstraktion

Abstraktion von der Demonstration um die Lösung der Aufgabe so allgemein wie möglich darzustellen.

→ Instanzen müssen falls möglich in Variablen umgewandelt werden; dabei muss sichergestellt werden, dass in der Ausführungsphase nur solche Variablen instantiiert werden, die räumliche Vorbedingungen erfüllen. Es wird also für jeden „generalisierten“ Operator ein Satz von Vorbedingungen in Form von wichtigen Relationen gespeichert. Generiert werden diese Vorbedingungen mit Hilfe von Hintergrundwissen und wiederum durch Rückfragen an den Benutzer

Man benötigt also eine deduktive Komponente, die die generalisierten Operatoren in Makro-Operatoren gruppieren. Da Makro-Operatoren weitere Makro-Operatoren als Kinder beinhalten können, lässt sich die gesamte Benutzervorführung auf verschiedenen Abstraktionsebenen darstellen. Durch die Generalisierung lässt sich die Lösung später auf ähnliche Problemklassen anwendbare Lösungsbeschreibungen abbilden. In dieser Phase lassen sich auch vom Demonstrator durchgeführte spontane und nicht zielorientierte Bewegungen ausfiltern.

Transfer/Abbildung auf Roboter & Szene

Transfer der internen Wissensrepräsentation auf das Zielsystem. Aus den zuvor gewonnenen semantischen Informationen lässt sich jetzt ein ausführbares Roboterprogramm generieren. Dafür müssen die Operationen auf die Operationen des Zielsystems abgebildet werden. Die Ausgabe dieser Phase ist eine Sequenz von Elementarbewegungen, die nur für das Zielsystem und das jeweilige Umweltmodell gültig sind. Diese kann direkt in das Simulationsmodul weitergeleitet werden.

Simulation

Simulation des physikalischen Vorgangs zur Validierung der getroffenen Entscheidungen. In der Simulation wird die gelernte Aufgabe von einem virtuellen Modell des Roboters in einer virtuellen Umwelt an virtuellen Objekten ausgeführt. Anhand einer visuellen Ausgabe lässt sich vom Benutzer die korrekte Ausführung der Aufgabe überprüfen.

Ausführung

Ausführung auf dem Zielsystem. Die zuvor validierte Sequenz elementarer Roboterbewegungen wird an den Roboter-Controller weitergereicht. Wenn bei der Modellierung in der Simulationsphase keine Fehler gemacht wurden, ist es sehr wahrscheinlich, dass die Ausführung nicht fehlschlagen wird.

3.3 Klassifikation von PdV-Verfahren

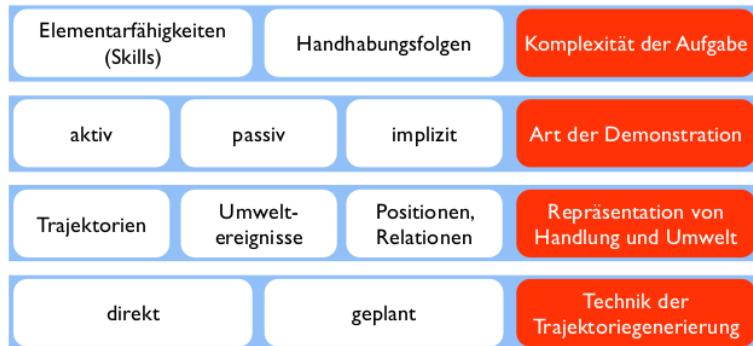


Abbildung 22: Klassifikationskriterien

Abbildung 22 zeigt die Klassifikationskriterien auf einen Blick.

Komplexität der Aufgabe

Siehe Tabelle 15

Art der Demonstration

Siehe Tabelle 16

¹Asada91, Koeppe95, Kaiser96, Ishikawa99, Bentivegna00, Calinon08, Pastor09

²Segre89, Inaba90, Kang94, Sagerer98, Aleotti06, Pardowitz07 Veeraraghavan08

³Friedrich98, Ikeuchi99, Inoue/Kuniyoshi94, Zöllner06

Elementarfähigkeiten	Komplexe Aufgaben
<p>Reflexe, Basisskills, einfache Bewegungen</p> <ul style="list-style-type: none"> • Lernen direkter Sensor-/Aktorzusammenhänge • Beispiele: Verwendung neuronaler Netze auf adaptive Regelkreise, Zustandsautomaten¹ <p>→ Probleme: Explizite Beispiele, stark konfigurationsabhängig, viele Trainingsbeispiele notwendig</p>	<p>Task, Montageaufgaben</p> <ul style="list-style-type: none"> • Interpretation von Handlungsfolgen² <p>→ Probleme: Breites Hintergrund-, Planungs- und Modellwissen, Klärungsdialoge mit dem Benutzern</p>

Tabelle 15: Kriterium 1 - Komplexität der Aufgabe

aktiv	passiv	implizit
<p>Aktive Beispiele</p> <ul style="list-style-type: none"> • Benutzer führt explizit vor • Beobachtung durch Sensorsystem (Datenhandschuh, Kameras)³ <p>→ Probleme: Aufwendige Sensorsysteme, Identifikation relevanter Aktionen und Ziele bzw. Zustände schwierig</p>	<p>Passive Beispiele</p> <ul style="list-style-type: none"> • Roboter wird durch externen „Master“ gesteuert, Signalaufzeichnung, Korrelation zwischen Sensor- und Aktordaten⁴ <p>→ Probleme: Gelerntes Wissen ist auf konkretes Zielsystem festgelegt</p>	<p>Implizite Beispiele</p> <ul style="list-style-type: none"> • Zielspezifikation durch Vorgabe graphischer Ikone⁵ <p>→ Probleme: Dialog umfangreich, Anpassung an Zielsystem</p>

Tabelle 16: Kriterium 2 - Art der Demonstration

Repräsentation von Handlung und Umwelt

Siehe Tabelle 17

Trajektorien	Umweltereignisse	Positionen, Relationen
<p>→ Problem: keine wesentliche Generalisierung (1:1 Abbildung)</p>	<p>Wirkungen und Reaktionen auf die Umgebung</p> <p>→ Probleme: Identifikation von Kausalitäten und Zustandsfolgen durch kognitive Operatoren</p>	<p>Objektlagen, Relationen und Operatoren</p> <p>→ Probleme: Beschränkung auf vorgegebenen Operatorumfang</p>

Tabelle 17: Kriterium 3 - Repräsentation von Handlung und Umwelt

Technik der Trajektoriengenerierung

Siehe Tabelle 18

⁴Kaiser96, Koeppe98, Billard07

⁵Takahashi/Ogata97, Sagerer98, Riepp97

⁶Kang97, Kaneko97

geplant	direkt
<p>Planung von Roboterbewegungen / Aktionen</p> <ul style="list-style-type: none"> • Planung erforderlich zur Berücksichtigung der Unterschiede in Demonstrations- und Ausführungsumgebung • Vollständige Nutzung der Leistungsfähigkeit des Robotersystems <p>→ Probleme: Umwelt- und Planungswissen erforderlich, intelligentes Planungssystem</p>	<p>Direkte Abbildung</p> <ul style="list-style-type: none"> • Explizites oder gelerntes Transformationsmodell⁶ <p>→ Probleme: Zielsystem und Zielumgebung müssen korrespondieren</p>

Tabelle 18: Kriterium 4 - Technik der Trajektoriengenerierung

3.4 Prozesskomponenten der interaktiven Programmierung

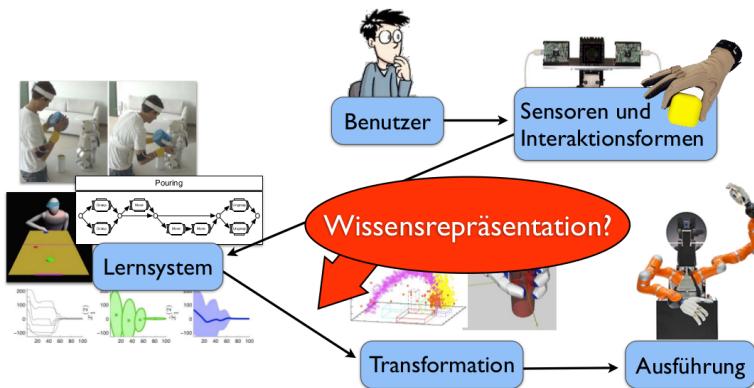


Abbildung 23: Interaktive Programmierung: Komponenten

Abbildung 23 skizziert die an PdV beteiligten Komponenten:

Benutzer und Mensch-Maschine-Interaktionsformen

Der Mensch, welcher die interaktive Programmierung vornimmt, kann multimodal mit dem Programmiersystem interagieren:

- **Physische Demonstration:** Hierbei vollzieht der Benutzer die Manipulationsaufgabe mit realen Objekten und wird dabei vom Programmiersystem durch Sensoren erfasst und seine Handlungen aufgezeichnet. Dies ist die natürlichste Art der Demonstration: Objekte können direkt mit den Händen oder mittels spezieller Vorführgeräte (z.B. Laserstift, 6D-Kugel) manipuliert werden. In jedem Fall ein großes Maß an Konzentration auf Seiten des Benutzers gefordert
- Nachteile:

- Er muss sich weiterhin über die Beschränkung und Konfiguration der ihn beobachtenden Sensoren bewusst sein (z.B. Verdeckung bei Beobachtung durch eine Kamera). Wegen der Abtastfrequenz der Sensoren ist auch die Geschwindigkeit, mit der die Demonstration vorgenommen wird, zu beachten. Oft ist es auch sinnvoll, die Manipulationsaufgabe in Teillösungen zu untergliedern, um gegebenenfalls Fehler leichter zu beheben.
- Aufgrund der beschränkten sensorischen, motorischen und intellektuellen Fähigkeiten des Menschen kann keine absolute Positioniergenauigkeit erwartet werden und auch die Wiederholgenauigkeit ist begrenzt.
- Auch produzieren menschliche Benutzer schon bei einfachen Manipulationsaufgaben häufig ineffiziente oder überflüssige Handlungen. Ebenfalls tragen der Umfang und die Komplexität der Demonstration der zu programmierenden Aufgabe dazu bei.

- **Graphische Demonstration:** Der Benutzer kann hierbei die Manipulationsaufgabe lösen, indem er 3D Objekte in einer simulierten Umgebung bewegt. Einige Probleme der physischen Demonstration mit Sensoren (eingeschränkte Beobachtbarkeit, limitierte Genauigkeit, zeitlicher Aufwand) sind bei dieser Art der interaktiven Programmierung nicht vorhanden. Es wird versucht, die Vorteile der intuitiven physischen Demonstration zu übernehmen und die angeprochenen Nachteile zu umgehen. Leider entstehen bei dieser Art von Demonstration andere Probleme. Auf einem Monitor ist die Wiedergabe einer 3D Szene mit ihren zu manipulierenden Objekten nur schwer zu erfassen. Sollen nur Translationen und Rotationen in einer Ebene ausgeführt werden (z.B Bestückung einer Platine) reicht ein Monitor aus. Besser eignen sich für die Darstellung von 3D Umgebungen Datenhelme und Shutter Brillen sowie 3D Höhlen. Mit den bisher beschriebenen Hilfsmitteln zur interaktiven Programmierung ist keine Möglichkeit der Kraftrückkopplung gegeben. Haptischen Ein- und Ausgabegeräte (Datenhandschuh mit Exoskelett und Phantom Manipulator) ermöglichen die Reaktionskräfte an den Benutzer weiterzugeben. Er kann besser auf die simulierte Welt einwirken, da er ein Feedback bekommt. Wegen der Echtzeitmodellaktualisierung ist aber ein hoher Rechenaufwand nötig.
Diese ist für den menschlichen Benutzer mit den gleichen Problemen verbunden wie die physische. Hinzukommen Probleme der Navigation und Koordination in der simulierten Umgebung. Aufgrund der fehlenden Realität der virtuellen Welt kann es sogar zu Simulationsübelkeit kommen. Dies geschieht durch den Widerspruch der Signale, welche vom Gleichgewichts-, Orientierungs-, Seh- und Gehörsinn aufgenommen werden.
- **Symbolische/Ikonische Demonstration:** Es wird eine Aktionssequenz erzeugt, durch graphisches Aneinanderreihen vorhandener Teillösungen. Für diese Art der Demonstration reicht eine menügesteuerte, konventionelle graphische Schnittstelle. Es müssen nur entsprechende Operatoren vorhanden sein, welche alle Informationen zur Manipulation einzelner Objekte beinhalten. Die Ressourcenanforderungen an das System sind dementsprechend gering. Bei ihr setzt der menschliche Benutzer aus vorhandenen Teillösungen eine Lösung für die Manipulationsaufgabe zusammen. Das Problem ist hierbei die Bestimmung der zu manipulierenden Objekte und die Parametrisierung der Bewegungsfolge. Die Kommentierung von Systemhypothesen, die Vermittlung der Sensorik von Aktionen und der eigenen Intention ist für den Benutzer bei entsprechenden Benutzerschnittstellen (symbolische, graphische Darstellung) leichter.
- **Kommentierung:** Sie ist eine ergänzende Interaktionsform zur physischen oder graphischen Programmierung. Der Benutzer nimmt hierbei keine aktive Rolle ein, sondern reagiert auf Systemhypothesen. Nach einer Demonstration erhält man die Möglichkeit, Systemhypothesen zu präsentieren und durch Auswahl, Editierungen und Ablehnung mit der Benutzerintension abzugleichen. In vielen Anwendungen (Textverarbeitung, Graphikanwendungen, Programmierung) sind textuelle oder menübasierte Schnittstellen ausreichend. In der Robotik kann es durch den räumlichen Bezug der Systeme zusätzlich nötig sein, über zweidimensionale graphische Interaktionsmuster hinaus eine dreidimensionale Schnittstelle zu bieten. Nur mit Kommentierung ist eine Überwachung und gegebenenfalls eine Korrektur der systeminternen Programmgenerierung möglich.

Sensoren

- **Bildgebende Sensoren:** meist Kameras; ermöglichen zusätzlich zur Aufnahme und Analyse der Demonstration eine Modellierung der Umwelt, kritisch hierbei ist der hohe Rechenaufwand und Schwierigkeiten für den Benutzer (muss im optimalen Bildbereich agieren und Verdeckungen zwischen Hand und Zielobjekt vermeiden)
- **Magnetfeldbasierte Positionssensoren:** direkte Bestimmung von Position und Orientierung der Benutzerhand; Erkennung nicht durch Verdeckungen und Lichtverhältnisse behindert; problematisch ist die quadratische Abnahme der magnetischen Feldstärke und Störungen durch metallische Gegenstände
- **Datenhandschuhe & -anzüge:** Dehmessstreifen und Lichtleiter; niedrige Genauigkeit

- **Exoskelette:** höhere Genauigkeit durch mechanische Struktur aber hohe Kosten, Gewicht, Einschränkung des Benutzers bei der Demonstration
 - **Interne Robotersensoren:** zuverlässige Werte (wenn für Demonstration und Ausführung der selbe Roboter verwendet wird ist bei gleicher Umweltsituation der Erfolg garantiert) aber hoher Geräteaufwand und geringer Komfort für Benutzer, daher eher bei Teach-In und Telerobotik verwendet
- Sensordatenfusion sinnvoll, aber ihrerseits mit Herausforderungen behaftet

Weltmodell sowie Planungs- und Entscheidungsmechanismen/ Programmiersystem

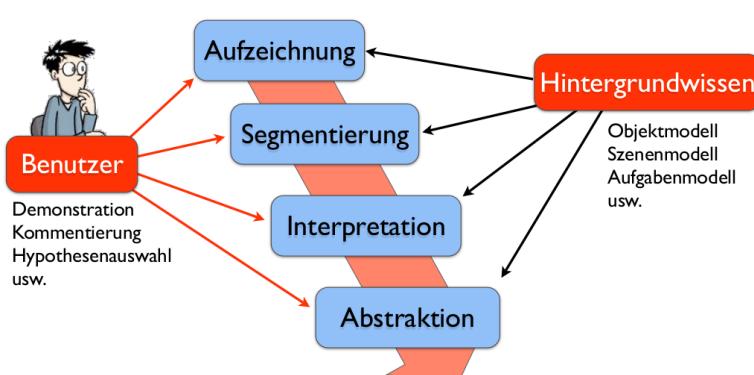


Abbildung 24: Lernsystem: Verarbeitungsschritte

Abbildung 24 zeigt die Bearbeitungsschritte eines interaktiven Programmiersystems

Ausführende Manipulatoren

Ein Manipulator ist ein Roboter, der sich aus einer Steuerung und mechanischen Komponenten (Manipulatorarm) zusammen setzt. Er ist in der Lage, die Position und Orientierung eines Objektes bezüglich eines Bezugskoordinatensystems zu ändern. Es sind zwei Arten der Wissensrepräsentation zu unterscheiden, wie in Tabelle 19 dargestellt.

3.5 Wissensrepräsentation: Beispielsysteme

Probabilistisch (Calinon & Billard)

⁷ Ziel: Lernen von Skills, z.B. Schachfigur bewegen

- Aktive, physische Demonstration am Roboter → kein Korrespondenzproblem
- Repräsentation durch Gaussian Mixture Models (GMM) → subsymbolisch
- Direkte Ausführung

Hierbei werden folgende Lerndaten benötigt:

- (θ, x, y, h) : n Demonstrationen mit je T Trajektorienpunkten
- θ : Gelenkwinkel des Roboters + Zeitstempel
- x : Kartesische Position der Hände + Zeitstempel
- y : Distanzvektor der Hände zur Startposition des Objekts + Zeitstempel

⁷[Calinon07]: What is the teacher's role in robot programming by demonstration?
On learning, representing and generalizing a task in a humanoid robot

Manipulatorabhängige Repräsentation: subsymbolisch	Manipulatorunabhängige Repräsentation: symbolisch
<p>Angabe von</p> <ul style="list-style-type: none"> • Aktionssequenz oder • Gelenkwinkel-, Kraft und Momenttrajektorien <p>Nachteile:</p> <ul style="list-style-type: none"> - bei Serviceanwendungen nicht flexibel genug, da invariante Trajektorien keine Veränderung der Lage zu den manipulierten Objekten in der Umwelt erlauben - Variationen in Material, Form und Gewicht beim Manipulationsobjekt können nicht berücksichtigt werden; statische Trajektorien verlieren bei einer Veränderung dieser Werte zur Programmausführung ihre Gültigkeit - unterschiedliche Manipulatoren weisen aufgrund der Kinematik der Manipulatorarme und Endeffektoren verschiedene Konfigurationsräume auf und besitzen verschiedene Sensorsausstattung - bei expliziter Trajektorienerfassung benötigt man viel Speicher <p>→ durch die schwach strukturierte Umwelt und die Forderung von Wiederverwendbarkeit im Dienstleistungsbereich ist die manipulatorabhängige Repräsentation hier nicht sinnvoll</p>	<p>Angabe von Sequenzen von Elementaroperatoren</p> <ul style="list-style-type: none"> • Elementaroperatoren sind Regelungen mit Start-, End- und Fehlerkriterien • Implementierung der Elementaroperatoren ist manipulatorunabhängig • Effekte in der Umwelt sind manipulatorunabhängig, Situation wird nur über lokale Sensorwerte erstellte Umweltinformation beschrieben <p>Bewertung:</p> <ul style="list-style-type: none"> + flexible Repräsentation und benutzerfreundliche symbolische Darstellung + größere Flexibilität gegenüber Umweltveränderungen und unterschiedlichen Manipulatortypen + Elementarfunktionen haben einen hohen Wiederverwendbarkeitswert, so braucht ein Programm nur die Sequenz der Elementarfähigkeiten enthalten und ihre einzelnen Parameter bestimmen und repräsentieren; oft weichen in einer dynamischen Welt die Parameter der Elementarfähigkeiten von denen der Benutzerdemonstration ab, daher werden für variable Parameter Berechnungsfunktionen und Auswahlbedingungen statt konkreter Werte benutzt - Elementarfähigkeiten müssen für jeden Manipulator a-priori erzeugt werden

Tabelle 19: Arten der Wissenrepräsentation

- h : Binärer Zustand des Greifers (offen, geschlossen) + Zeitstempel

Das Lernverfahren besteht dann aus folgenden Schritten:

- Dimensionsreduktion und zeitliche Angleichung: $(\theta, x, y, h) \rightarrow (\theta', x', y', h')$
- Lernen eines GMMs (vgl. Abbildung 25) für alle Komponenten, z.B. x' (4-dimensional) mit Dichte $p(x')$:

$$p(x') = \sum_{i=1}^k p(i)p(x'|i) = \sum_{i=1}^k \pi_i \mathcal{N}(x'; \mu_i, \Sigma_i)$$

mit k = Anzahl der Normalverteilungen, π = Gewichtung, \mathcal{N} = Normalverteilung

- Bestimmung von k : Bayes'sches Informationskriterium und EM-Algorithmus

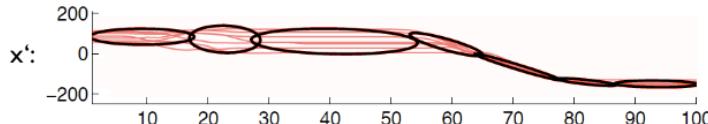


Abbildung 25: Gaussian Mixture Model

Die Repräsentation gestaltet sich folgendermaßen:

- Probabilistische Darstellung der Trajektorien $x' : t \mapsto \mathcal{N}(\mu_{x'}(t), \Sigma_x(t))$
- Berechnung durch Gaussian Mixture Regression (vgl. Abbildung 26): Gewichtung der bedingten Wahrscheinlichkeiten $p(x', i|t)$

$$\mu_{x'}(t) = \sum_{i=1}^k \beta_i(t) \mu_{i,x'|t} \text{ und } \Sigma_{x'}(t) = \sum_{i=1}^k \beta_i(t)^2 \Sigma_{i,x'|t}$$

mit $p(x', i|t) = \mathcal{N}(\mu_{i,x'|t}, \Sigma_{i,x'|t})$ und $\beta_i(t) = \frac{p(t|i)}{\sum_{j=1}^k p(t|j)}$

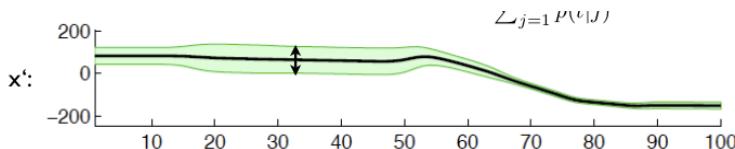


Abbildung 26: Gaussian Mixture Regression

Die Ausführung besteht dann aus folgenden Schritten:

- Definition eines quadratischen Ähnlichkeitsmaßes: „metric of imitation“
- Gewichtung der Abweichung von den Mittelwerten in t , z.B. $\mu_{x'}(t)$
- Wahl der Gewichtsmatrix: $(\Sigma_{x'}(t))^{-1}$
- Bestimmung des Nachfolgerzustands $\theta(t+1)$ bzw. der Transition $\theta(t) \rightarrow \theta(t+1)$, die das Ähnlichkeitsmaß minimiert
- Jacobi-Matrix zur Kombination von kartesischen und Gelenkwinkel Einschränkungen

Eine Bewertung des Verfahrens ist in Tabelle 20 dargestellt.

Vorteile	Nachteile
<ul style="list-style-type: none"> + schnelles Verfahren, ähnlich Playbackprogrammierung + automatische Adaptierung an Änderungen der Objektpositionen 	<ul style="list-style-type: none"> - relevante Merkmale manuell definiert, hier z.B. nur Distanz zu Startposition - geringe Generalisierung, da keine Vorbedingungen, Ziele, Kollisionen keine zielgerichtete Erzeugung von Bewegungen - keine Validierung

Tabelle 20: Zusammenfassung: Calinon, Billard

Dynamisch (Pastor & Schaal)⁸

Ziel: Lernen von Skills, z.B. Tennisschwung

- Aktive, physische Demonstration am Roboter → kein Korrespondenzproblem
- Repräsentation durch Dynamic Movement Primitives (Differentialgleichungen)
- Direkte Ausführung

Die Repräsentation ist wie folgt:

- Implizite Darstellung durch Menge von Differentialgleichungen:

$$\begin{aligned}\tau \dot{v} &= K(g - x) - Dv + (g - x_0)f \\ \tau \dot{x} &= v\end{aligned}$$

- x = Position, v = Geschwindigkeit, K = Federkonstante, D = Dämpfung, g = Ziel, x_0 = Start, τ = zeitliche Skalierung
- f = nicht-lineare Funktion, die die Demonstrationsmenge approximiert:

$$f(s) = \frac{\sum_{i=1}^n w_i \psi_i(s)s}{\sum_{i=1}^n \psi_i(s)} \text{ mit } \tau \dot{s} = -\alpha s$$

- Vorteil: Gewichte hängen nicht von τ, x_0 und g ab
- Änderungen von Start, Ziel und der zeitlichen Skalierung möglich
- Generalisierung eingeschränkt möglich

Das Lernen läuft in folgenden Schritten ab:

- Berechnung von $v(t), \dot{v}(t)$ für jede Demonstration $x(t)$
- $s(t)$ wird durch Integration berechnet
- Der Wert $f(s)$ wird berechnet
- ψ_i sind nicht normalisierte Normalverteilungen („Gauß'sche Basisfunktionen“)
- Bestimmung der Parameter w_i durch lineare Regression

Die Ausführung erfolgt über die Berechnung von $v(t), \dot{v}(t)$ im aktuellen Zustand und Integration. Eine Bewertung des Verfahrens ist in Tabelle 21 dargestellt.

Vorteile	Nachteile
<ul style="list-style-type: none"> + schnelles Verfahren + automatische Adaptierung an Start und Ziel + lokale Hindernisvermeidung möglich 	<ul style="list-style-type: none"> - relevante Merkmale manuell definiert - geringe Generalisierung - keine Validierung

Tabelle 21: Zusammenfassung: Pastor, Schaal

⁸[Pastor09]: Learning and generalization of motor skills by learning from demonstration

Sub-/Symbolisch (IPoR II)

Interaktives Programmieren von Robotern (IPoR) wurde an der Universität Karlsruhe entwickelt.
Problembeschreibung:

- Robotersystem mit sehr vielen Freiheitsgraden (Kuka LBR / SAH/HIT: 40)
- Fingerfertige Manipulation fester Körper („rigid bodies“)
- Roboterarbeitsraum in realen Situationen stark eingeschränkt (z.B. Kollisionen)
- „Korrespondenzproblem“: Mensch und Roboter haben unterschiedliche Kinematik, d.h. keine direkte Abbildung menschlicher Bewegungen möglich

Einsatz von Planungsmethoden:

- Repräsentation der Manipulationsaufgabe als Bahnplanungsproblem mit Einschränkungen
- Autonome Planung von Bewegungen, die das Ziel einer Manipulationsaufgabe erfüllen
- Problem: Manuelle Definition des Planungsproblems ist komplex (z.B. ≤ 40 dofs)

→ Lernen von Planungsproblemen aus der Beobachtung des Menschen

Repräsentation als Bahnplanungsproblem mit Einschränkungen:

- Beschränkung der Bewegung eines Koordinatensystems relativ zu einem zweiten Koordinatensystem (ähnlich „Task Frames“)
- Drei Typen von Bewegungseinschränkungen:
 - Positionseinschränkungen
 - Orientierungseinschränkungen
 - Richtungseinschränkungen
- Definition: Eine Einschränkung ist ein 5-Tupel (t, f, M, g, R) mit
 - Typ t
 - Koordinatensystemen f, g
 - homogener Transformationsmatrix M , relativ zu f
 - Region R
- Wann ist eine Einschränkung erfüllt?
 - Transformation von M definiert in f relativ zu g :
$$M' = {}^0 H_g^{-1} \cdot {}^0 H_f \cdot M$$

$$M' = {}^g H_f \cdot M$$
 - Umwandlung von M' in 3D-Vektor m' :
 - t = Position, Richtung: $m' = (xyz)$
 - t = Orientierung: $m' = (r_x r_y r_z)$
 - Bestimmung des nächsten Punkts n in R und der Distanz $d = |m' - n|$
 - Erfüllt, wenn $d < \varepsilon$
- Repräsentation des Planungsproblems als „Strategiegraph“: Tupel $X, C_n^t, C_e^t, C_n^b, C_e^b$ mit Knoten X , zeitliche Einschränkungen der Knoten C_n^t und Kanten C_e^t , Bewegungseinschränkungen der Knoten C_n^b und Kanten C_e^b

Abbildung 27 zeigt ein Beispiel eines solchen Graphen.

Im Folgenden soll der PdV-Zyklus (vgl. Abbildung 21) beispielhaft an IPoR II verdeutlicht werden.

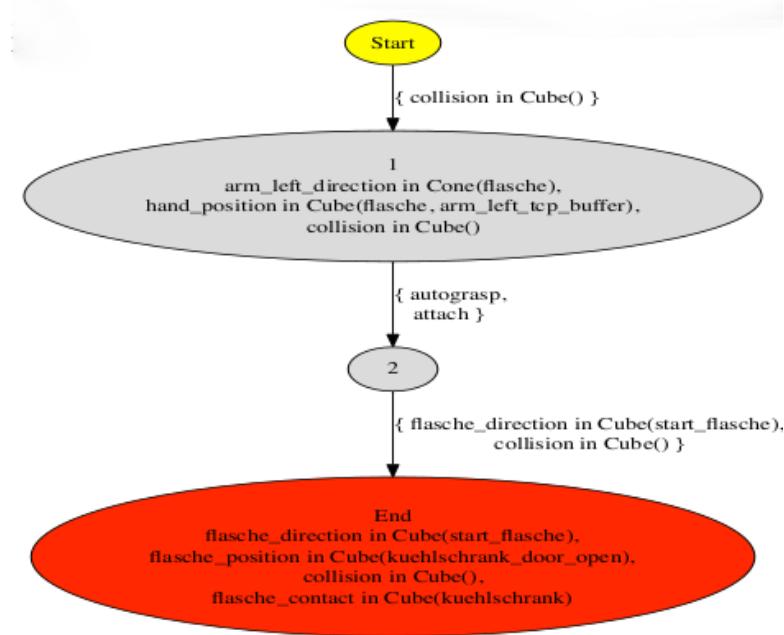


Abbildung 27: Beispiel eines Strategiegraphen: „Einschenken“ → Die Ausführungsumgebung bezieht sich auf Unterschiede in Objekten, Objektposen und Hindernissen (verglichen mit Demonstrationen). Einschränkungen im Strategiegraphen referenzieren Koordinatensysteme. Deren Lage wird automatisch adaptiert. Bei Ausführung unter Verwendung von Bahnplanung repräsentieren die Knoten des Graphen Teilziele der Manipulationsaufgabe, seine Kanten Übergänge zwischen den Teilzielen. Ein Planungsproblems wird für jedes Teilziel gelöst.

Beobachtung Als Sensorik wird verwendet: Mikrofon, Deckenkameras, Stereokamera mit Pan-Tilt-Unit, Flock-of-Birds, Voodoo, Cybergloves

Virtual Technology - Datenhandschuh, Meßprinzip: Dehnmessstreifen, 20 Fingerbeugungs- und Spreizwinkel + 2 Freiheitsgrade im Handgelenk

Bestimmung der Position und Orientierung der menschlichen Hände sowie von Objekten: Magnetfeld-basierter Positionstracker, Stereokamera

Segmentierung

- Ziel (mit Interpretationsphase): Repräsentation der demonstrierten Handlung durch Sequenz von zu erfüllenden Teilzielen (= Topologie des Strategiegraphs)
- Ansatz: schwwellwertbasierte Segmentierung zur Bestimmung von markanten Zeitpunkten der Demonstration
- Vorteile: einfache Interaktionsmöglichkeit während der Demonstration, einfache Korrektur von Hypothesen
- Erzeugung eines Segmentierungspunkts, wenn Hand-, Fingergeschwindigkeit gering ist und mindestens ein Finger Objektkontakt hat

Interpretation

- Klassifikation der Segmentierungspunkte in 4 Typen auf Basis des Weltzustands an den Intervallgrenzen:
 - Kein Objekt
 - Objekt aufgenommen

- Objekt gehalten
- Objekt losgelassen

- Lernen des Planungsmodells

- Topologie des Strategiegraphs: Segmentierung der Bewegungen des linken und rechten Arms, Kombination zu einem Graphen (vgl. Abbildung 28a)
- Erzeugung der Bewegungseinschränkungen: Manuell definierte Koordinatensysteme für alle Objekte: $K = \{\text{Flaschenöffnung}, \text{Becheröffnung}, \text{Rechter Zeigefinger}, \text{Welt}, \dots\}$; Erzeugung aller möglichen Einschränkungen (t, f, M, g, R) mit $f, g \in K$, Typ t beliebig, für jeden Knoten und jede Kante \rightarrow Region R muss bestimmt werden (vgl. Abbildung 28b)
- Beispiel: $f = \text{Flaschenöffnung}$, $g = \text{Welt}$, $t = \text{Richtung}$; $f = \text{Flaschenöffnung}$, $g = \text{Becheröffnung}$, $t = \text{Position}$

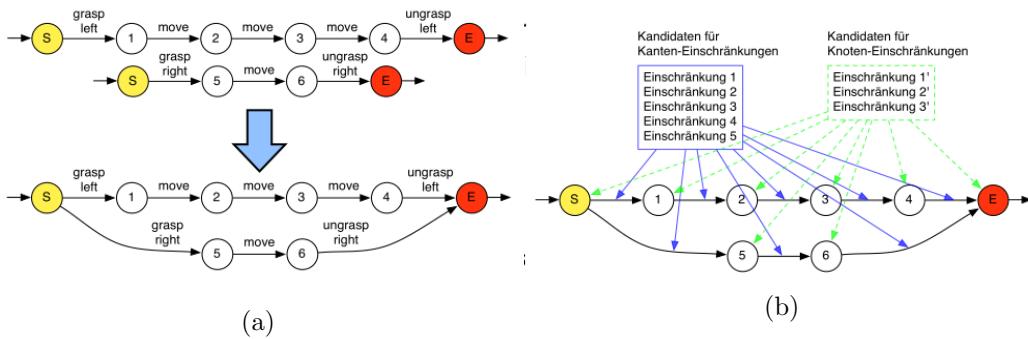


Abbildung 28

- Bestimmung der Region R : Für jede Einschränkung (t, f, M, g, R)

- Bestimmung des Werts von f in jedem Punkt t_i des Segments: $g_f^H(t_i) \cdot M$
 - Umwandlung in 3D-Vektor $m'(t_i)$
 - Ergebnis: Menge von 3D-Vektoren
 - Bestimme Region R , die alle 3D-Vektoren einschließt
- Beispiel: Knoten-Einschränkung ($f = \text{Flaschenöffnung}$, $g = \text{Becheröffnung}$, $t = \text{Position}$)

Abstraktion Ziel: Anwendbarkeit des Planungsmodells in neuen Situationen auf verschiedenen Robotern

- Weitere Einschränkungen: Kräfte, Kontakte, Objektbewegung
 - Wesentliche Abstraktion durch Koordinatensysteme und Einschränkungen
- Abbildung von objektabhängigen Koordinatensystemen, z.B. Flasche.Öffnung → Milchpackung.Öffnung

Ausführung

- Gelernte Einschränkungen definieren Suchraum für Roboterbewegungen
- Einsatz von *Bahnplanung unter Einschränkungen* zur Ausführung von gelernten Planungsmodellen
- Einsatz von *Griffplanung* zur Bestimmung qualitativ hochwertiger Griffe

Eine Bewertung des Verfahrens ist in Tabelle 22 dargestellt.

Vorteile	Nachteile
<ul style="list-style-type: none"> + Generalisierung auf Basis von Objekteigenschaften + Start- und Zielbeschreibung, Validierbarkeit + Hindernisvermeidung und Berücksichtigung von Einschränkungen + mehrere Lösungen und beliebige Optimalitätskriterien 	<ul style="list-style-type: none"> - hoher Aufwand (Planungszeit, Simulationszeit) - 3d-Modelle der Objekte, menschlichen Hand notwendig - automatische Segmentierung bei dynamischen Bewegungen schwierig

Tabelle 22: Zusammenfassung: IPOR II

3.6 Bahnplanung

Zunächst einige relevante Definitionen:

- **Konfiguration K :** vollständige, eindeutige Beschreibung des Zustands eines Roboters A , z.B.
 - im euklidischen Raum durch seine Position und Orientierung
 - im Gelenkwinkelzustandsraum durch die Werte der Gelenke
- **Konfigurationsraum \mathbb{K} :** Raum aller möglichen Konfigurationen des Roboters A
- **Weg** des Roboters A von der Konfiguration K_{Start} zu K_{Ziel} ist eine stetige Abbildung:

$$\begin{aligned}\tau : [0, 1] &\rightarrow \mathbb{K} \\ \tau(0) &= K_{Start}, \tau(1) = K_{Ziel}\end{aligned}$$

- **Einschränkung** für den Roboter A ist eine Abbildung:

$$\lambda : \mathbb{K} \rightarrow [0, 1]$$

- **Arbeitsraumhindernis H :** Raum, welcher von einem Objekt im Arbeitsraum eingenommen wird
- **Konfigurationsraumhindernis \mathbb{K}_H :** Menge aller Punkte des Konfigurationsraumes, welche innerhalb eines Arbeitsraumhindernisses H_i liegen:

$$\mathbb{K}_{H_i} = K \in \mathbb{K} \mid K \in H_i$$

- **Hindernisraum \mathbb{K}_H :** Menge aller Konfigurationsraumhindernisse:

$$\mathbb{K}_H = \bigcup_i \mathbb{K}_{H_i}$$

- **Freiraum \mathbb{K}_F :** Menge aller Punkte aus \mathbb{K} , welche nicht im Hindernisraum liegen:

$$\mathbb{K}_F = \mathbb{K} \setminus \mathbb{K}_H$$

- **kollisionsfreier Weg τ :** Weg mit $Bild(\tau) \subseteq \mathbb{K}_F$, also ein Pfad welcher alle Einschränkungen erfüllt

Bei einer Bahnplanung im Konfigurationsraum werden Bewegungen eines Roboters als **Trajektorie im Konfigurationsraum**, d.h. als Zustandsänderungen über die Zeit relativ zu einem stationären Koordinatensystem (kartesischer Raum, Gelenkwinkelraum) aufgefasst:

- Gegeben: K_{Start} = Startkonfiguration, \wedge_{Ziel} = Menge der Zieleinschränkungen
- Gesucht: Kollisionsfreier Weg τ von K_{Start} nach K mit $\lambda(K) = 1 \forall \lambda \in \wedge_{Ziel}$
- Bedingungen: i.A. Gütekriterien, Neben-, Rand- sowie Zwangsbedingungen

Hierbei sei angemerkt, dass von Roboter und Umwelt zu einem Punkt im Konfigurationsraum abstrahiert wird. Die Kollisions- bzw. Einschränkungsüberprüfung stellt eine Blackbox

$$f : \mathbb{K} \rightarrow \{0, 1\}$$

Beispiel: $f(K) = \bigwedge_{v \in \wedge_{Weg}} v(K) \geq \varepsilon(v)$

dar. Bei Entwicklung allgemeiner Planungsverfahren auf Basis dieser Abstraktion entspricht die Bahnplanung einer Suche nach einer stetigen Verbindung zweier Punkte im Konfigurationsraum. Eine explizite Beschreibung des Freiraums ist nicht notwendig, d.h. das Suchverfahren ist unabhängig von der Struktur und Repräsentation des Freiraums.

Simpler Rapidly-exploring Random Tree (RRT) Planer

RRT-Algorithmus Wie bereits zu Anfang des Kapitels erwähnt, bringt die humanoide Servicerobotik zahlreiche Anforderungen mit sich, sowie die Manipulation beliebiger Objekte, das selbstständige Lösen komplexer Aufgaben und den Einsatz im menschlichen Umfeld, d.h. humanoide Serviceroboter müssen in einer sehr komplexen Umgebung agieren und verfügen über sehr viele Freiheitsgrade (der Roboter Albert II beispielsweise hat 13df, 6 im Arm, 3 in der Plattform und 4 in der Hand. Daraus ergibt sich ein 13-dimensionaler Konfigurationsraum \mathbb{R}^{13} als reellwertige Grundlage). Ein Bahnplanungsalgorithmus, der hiermit umgehen kann ist der RRT = Rapidly-exploring Random Tree⁹, welcher zur effizienten Durchsuchung hoch-dimensionaler Räume entwickelt wurde. Er ist geeignet für holonome und nicht-holonome Problemstellungen mit Einschränkungen. Es wird inkrementell eine Baumstruktur aufgebaut und dabei der erwartete Abstand eines Punkts zu einem Knoten im Baum minimiert. Wenn die Zeit t gegen unendlich geht kommt man beliebig nah an jeden beliebigen Punkt. Der Algorithmus erreicht eine hohe Geschwindigkeit durch schnelles Wachstum in nicht explorierte Bereiche. Die Wurzel ist ein Punkt im 13-dimensionalen Konfigurationsraum. Pseudocode ist in Algorithmus 1. Eine graphische Veranschaulichung zeigt Abbildung 36. Der Knoten mit der größten Voronoi-Region hat jeweils die größte Wahrscheinlichkeit, als nächstes erweitert zu werden (**Voronoi-Bias**). Da am Anfang die Voronoi-Gebiete am Randbereich groß sind findet zunächst eine rasche Exploration und dann eine Verfeinerung statt (vgl. Abbildungen 30 und 31).

Algorithm 1 RRT

```
BUILD_RRT( $K_{Start}, n, \varepsilon$ )
1:  $T.init(K_{Start})$                                 ▷ Neuer Baum mit Startkonfiguration in der Wurzel
2: for  $k = 1$  to  $n$  do
3:    $K_{Zuf} \leftarrow \text{RAND\_CONF}()$            ▷ Gleichverteilt zufällige Erzeugung einer Konfiguration
4:    $K_{Nahe} \leftarrow \text{NEAREST\_VERTEX}(K_{Zuf}, T)$     ▷ Bestimmung des nächsten Knotens
5:    $K_{Neu} \leftarrow \text{EXTEND}(K_{Nahe}, K_{Zuf}, \varepsilon)$     ▷ Erzeugung einer neuen Konfiguration
6:    $T.add\_vertex(K_{Neu})$ 
7:    $T.add\_edge(K_{Nahe}, K_{Neu})$ 
8: end for
9: return  $T$ 
```

RRT: Zusammenfassung

- Allgemeines Verfahren zur Durchsuchung hoch-dim. Räume
- Online-Verfahren

⁹[LaValle/Kuffner99]: Randomized Kinodynamic Planning

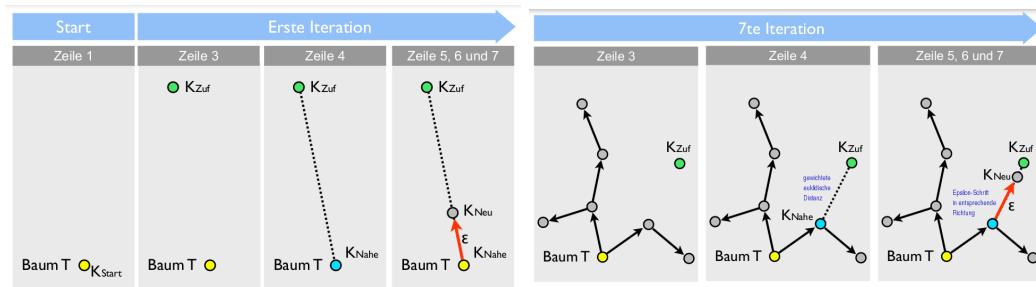


Abbildung 29: Graphische Veranschaulichung - RRT

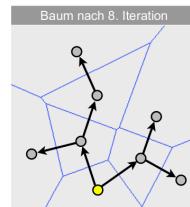


Abbildung 30: Voronoi-Bias – Detail

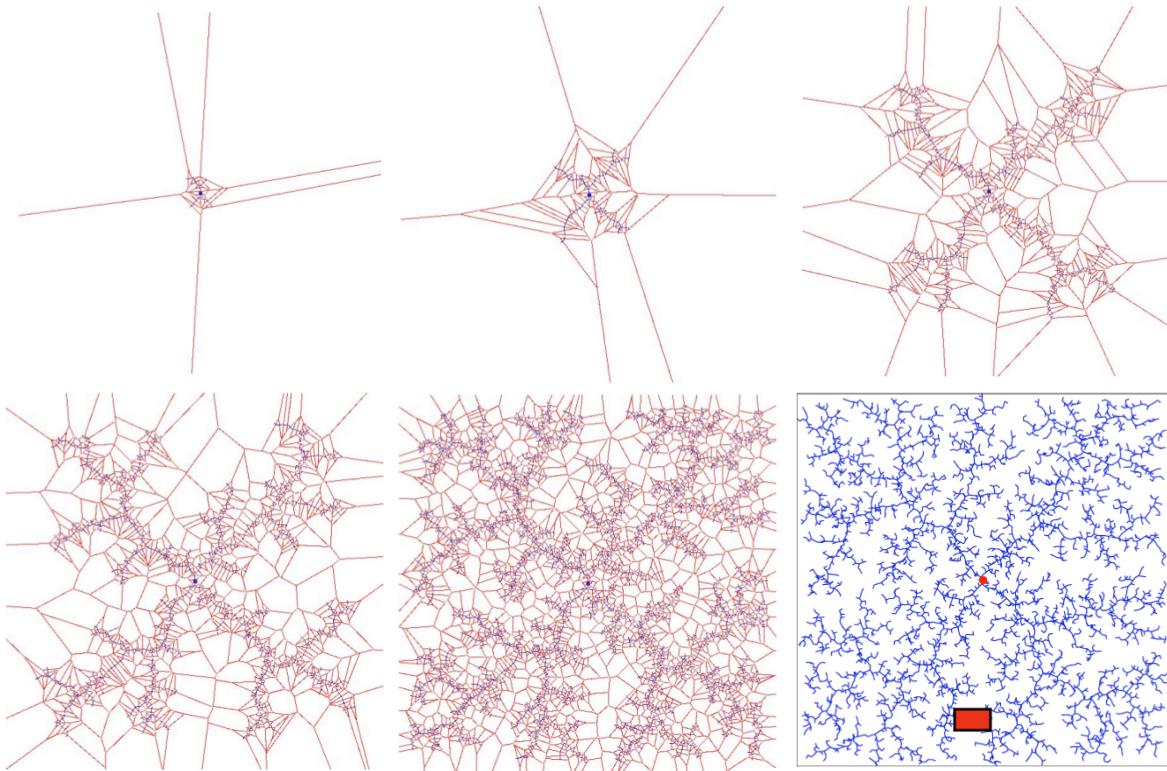


Abbildung 31: Voronoi-Bias

- Approximation des Suchraums durch eindimensionale Struktur: Baum
- Rasche Exploration des Suchraums: Voronoi-Bias
- Probabilistische (oder deterministische) Stichprobenerzeugung
- Einfach zu implementieren, nur wenige Parameter (ε , Distanzfunktion auf \mathbb{K})

Anwendung in der Bahnplanung Es sind drei Fragen zu klären:

1. Wie Einschränkungen berücksichtigen?
2. Wie zielgerichtet suchen?
3. Wie kollisionsfreie Wege erzeugen?

Hierzu muss der Basisalgorithmus erweitert werden:

- Es werden nur Konfigurationen hinzugefügt, die alle Einschränkungen erfüllen.
- Bei der Stichprobenerzeugung werden mit einer bestimmten Wahrscheinlichkeit Zielkonfigurationen generiert, der Baum wächst in die Richtung der Zielkonfigurationen.
- Der Planungsprozess ist beendet, wenn die letzte Konfiguration die Zieleinschränkungen erfüllt.

Frage 1 & 2: Einbeziehung von Einschränkungen und Umgang mit Kollisionen (vgl. Abbildung 32)

- K_{Neu} wird übernommen, wenn alle Einschränkungen zwischen K_{Nahe} nach K_{Neu} erfüllt sind
- Keine Distanz, nur Ja / Nein
- Kollisionen auf Grundlage eines Geometriemodells der Objekte (Standard: 3D-Dreiecksnetze)
- Kollisionsüberprüfung: mindestens zwei Dreiecke schneiden sich (oder 1. Objekt in 2. komplett enthalten)
- hierzu gibt es optimierte Algorithmen, ist vergleichsweise langsam

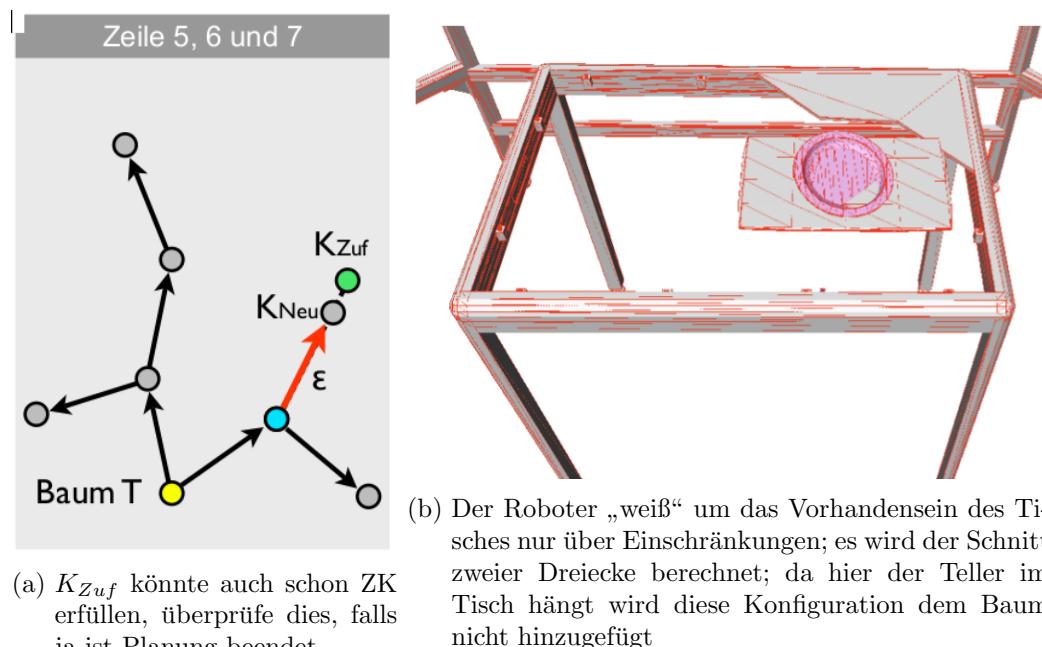


Abbildung 32: Einbeziehung von Einschränkungen und Kollisionen

Frage 3: Wie zielgerichtet planen? Wie in Abbildung 31 dargestellt definiert die Menge der Zieleinschränkungen i.d.R. ein stark begrenztes Gebiet in \mathbb{K} (roter Bereich im 6. Panel der Abbildung), z.B. endliche Menge von gültigen Zielkonfigurationen. Somit ist die Wahrscheinlichkeit eine Zielkonfiguration zufällig zu erzeugen minimal, es dauert recht lange bis der Baum in den betreffenden Bereich reinwächst. Daher ist die in Algorithmus 2 dargestellte Modifikation der Stichprobenerzeugung sinnvoll (vgl. Abbildung 33). Der resultierende Algorithmus ist Algorithmus 3.

Algorithm 2 Detail

```

    RAND_CONF( $f_{Ziel}, \delta$ ) ▷ z.B.  $\delta = 0.01$ 
1:  $P \sim U([0, 1])$ 
2: if  $P < \delta$  then
3:   return  $K \in \mathbb{K} : f_{Ziel}(K) = 1$ 
4: else
5:   return  $K \sim U(\mathbb{K})$ 
6: end if

```

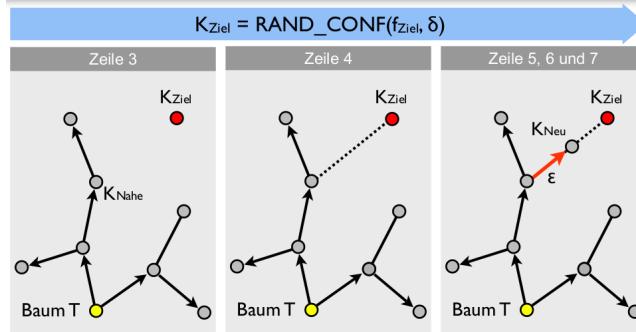


Abbildung 33: Erweiterung – Zielgerichtet planen mit RRT

Algorithm 3 Simpler RRT Planer

```

RRT_SIMPLE( $K_{Start}, f_{Ziel}, f_{Weg}, \varepsilon, \delta$ )
1:  $T.\text{init}(K_{Start})$  ▷  $f_{Ziel}$  ist Blackbox für Zieleinschränkungen
2: for  $k = 1$  to  $MAX$  do
3:    $K_{Zuf} \leftarrow \text{RAND\_CONF}(f_{Ziel}, \delta)$ 
4:    $K_{Nahe} \leftarrow \text{NEAREST\_VERTEX}(K_{Zuf}, T)$  ▷  $f_{Weg}$  ist Blackbox für Einschränkungen
5:    $K_{Neu} \leftarrow \text{EXTEND}(K_{Nahe}, K_{Zuf}, \varepsilon)$  ▷ Erzeugung einer neuen Konfiguration
6:   if  $f_{Weg}(K_{Neu}) = 1$  then
7:      $T.\text{add\_vertex}(K_{Neu})$ 
8:      $T.\text{add\_edge}(K_{Nahe}, K_{Neu})$ 
9:     if  $f_{Ziel}(K_{Neu}) = 1$  then ▷ Lösungspfad: Rückverfolgung der Elternknoten beginnend mit dem letzten Knoten
      dem letzten Knoten
10:    return  $T$ , Gefunden
11:  end if
12: end if
13: end for
14: return  $T$ , Nicht gefunden

```

Beispiel für die Anwendung des Simpelen RRT Planers: Holonomic Planung für einen Roboterarm mit 7 Freiheitsgraden (vgl. Abbildung 34)

- Konfigurationsraum $\mathbb{K} = [0, 1]$ (Gelenkwinkelbereich $[-170^\circ, +170^\circ]$ wird auf Bereich $[0, 1]$ normiert)
- Distanzfunktion auf \mathbb{K} : $d(k, s) = \sum_{i=1 \dots 7} w_i (k_i - s_i)$
- Gewichtsvektor $w = (1.5, 1.5, 1.5, 1.25, 1, 1, 1)$
- $\varepsilon = 0.001, \delta = 0.01$ (Schrittweite: mit welcher WK man Zielkonfiguration erzeugt)
- $f_{Ziel}(K) = 1 \Leftrightarrow K = K_{Ziel}$
- $f_{Weg}(K) = 1 \Leftrightarrow K$ ist kollisionsfrei

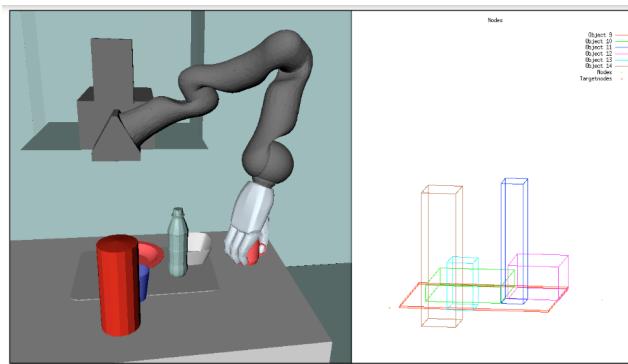


Abbildung 34: Beispiel für die Anwendung des Simplen RRT Planers: Arm bewegt sich relativ weich, auf recht glatter Bahn

Pfadglättung und Hinderniserweiterung

Reale Anwendung: Der Planer kann so nicht direkt eingesetzt werden

- Glättung notwendig, da Weg nicht glatt
- Hinderniserweiterung notwendig, da kleine Abweichungen von der Bahn zu Kollisionen führen können

Verringerung der Länge des Lösungswegs (siehe Abbildung 35):

- Zufällige Wahl zweier Knoten im Lösungsweg
- Überprüfung der Verbindung beider Knoten auf Einhaltung der Einschränkungen
- Falls die Verbindung geringere Kosten aufweist als der entsprechende Teillösungspfad, Verbindung der beiden Knoten und Löschen der dazwischenliegenden Knoten aus dem Lösungspfad

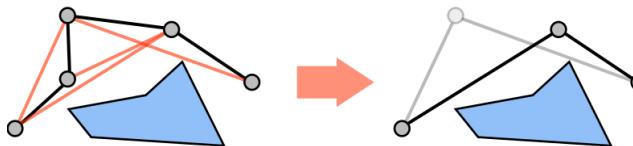


Abbildung 35: rot – gültige Abkürzungen → Kollisionsfreiheit muss erhalten bleiben; normaler Lösungspfad hat etwa 1000 Knoten

Hinderniserweiterung zur Berücksichtigung eines Toleranzbereichs im Planungsprozess durch Vergrößerung der Hindernisse

- Gründe:
 - Diskretisierung des Konfigurationsraums bzw. Abtastung von Trajektorien
 - Fehler der Objektlokalisierung: Kalibrierung des Roboters, Objekterkennung/-verfolgung
 - Abweichungen in der Bewegung: Rutschen, lockere Seilzüge, Modellierung, Impedanzregelung, Interpolationsart
 - Padding: Rand um beliebiges Hindernis festlegen (oft vorhandener Parameter in State-of-the-Art Bahnplanungs-Bibliotheken), vor allem für dünne Gegenstände sinnvoll
- Probleme:
 - Künstliche Einschränkung des Arbeitsraums
 - ε muss klein gewählt werden

- Annahme: Wenn der Roboter an einem Punkt hindernisfrei ist und 0.5mm weiter auch, dann auch dazwischen
- Kann zu Problemen führen; kontinuierliche Kollisionserkennung als Workaround, Distanzberechnung erlaubt exakte Bestimmung aber super aufwendig
- Erzeugung von Narrow-Passages: Lücken dicht gemacht durch Hindernisvergrößerung, eventuell keine gültige Lösung mehr vorhanden

Zusammenfassung – Simpler RRT Planer:

- Probabilistisch vollständig
 - Uniforme Stichprobenverteilung → Bereiche im Suchraum mit geringem Lebesgue-Maß, z.B. Narrow Passages, werden nicht abgedeckt
 - Hohe Laufzeitvarianz
 - Kein problemspezifisches Wissen
 - Ineffiziente Ergebnispfade → Glättung notwendig
- ⇒ **Manipulation:** Einschränkungen können Unterräume mit Lebesgue-Maß 0 erzeugen
- Modifikation notwendig
 - Blackbox-Formulierung der Einschränkungen problematisch

TC-RRT: Planung mit Task Constraints

Zweck: Berücksichtigung von Einschränkungen im Planungsprozess
Wiederholung:

- Ziele haben gleiche Struktur wie Einschränkungen
 - Kollisionsfreiheit ist eine Einschränkung
 - Erweiterung des Blackbox-Konzepts auf Einschränkungen möglich.
Problem: Narrow-Passages und Nullmengen treten sehr viel häufiger als bei Kollisionen auf.
- spezielle Methoden zur Berücksichtigung von Einschränkungen nötig

Ansatz¹⁰:

- Wesentliches Konzept: Erweiterung der Einschränkung-Definition um Richtungsfunktion im Taskraum („Öffnen der Blackbox“)
- Richtungsfunktion berechnet den Abstand und den Richtungsvektor zu einem Zustand, der die Einschränkung erfüllt
- Verwendung lokaler Optimierung mit Gradientenabstieg um von beliebiger Konfiguration zu einer gültigen Konfiguration zu kommen
- Funktioniert sehr gut in der Praxis (oftmals einfache, konvexe Einschränkungen)
- Constraint-Definition durch Angabe der Richtungsfunktion: $\lambda : \mathbb{K} \rightarrow \mathbb{K}_{Kart} \times \mathbb{R}$
 - \mathbb{K}_{Kart} ist der kartesische Arbeitsraum (z.B. $\mathbb{R}^3, \mathbb{R}^6$)
 - Hier: $\lambda(K) = (x \ y \ z \ rx \ ry \ rz \ a))$ mit rx, ry, rz skalierte Rotationsachsen
- Ersetzung der Extend-Funktion durch ConstrainedExtend

¹⁰basiert auf [Stilman07], [Berenson09]

- ConstrainedExtend berechnet neue Konfiguration, die alle Einschränkungen erfüllt
- 3 Methoden zur Berechnung der ConstrainedExtend-Funktion:
 - Randomized Gradient Descent (RGD)
 - First Order Retraction (FR)
 - Tangent Space Sampling (TS)

Randomized Gradient Descent (RGD) (vgl. Abbildung 36a):

- Toleranzwert für Einschränkungen: α
- Zufällige Bestimmung von n Nachbarn von K_S (in Hyperkugel mit Radius d_{max})
- Falls minimale Distanz der Nachbarn kleiner als die Distanz von K_S , ersetze K_S mit dem Nachbarn mit kleinster Distanz
- Wiederholen bis maximale Iterationszahl erreicht oder die Distanz von $K_S < \alpha$ ist
 \Rightarrow Keine Richtungsinformation notwendig (einige gültige Stichproben liegen in Ebene)

First Order Retraction (FR) (vgl. Abbildung 36b):

- Toleranzwert für Einschränkungen: α
- Speichern von K_S in K_O
- Berechnen der Distanz Δx von K_S
- „Einfahren“ von K_S : $K_S = K_S - J(K_S)^\dagger \Delta x$ (Bemerkung: die Notation A^\dagger bezeichnet die Matrix, die durch Transponierung und Konjugation einer gegebenen komplexen Matrix A entsteht)
- Iterieren bis die Distanz $< \alpha$ oder Abstand $|K_O - K_S|$ größer als $|K_O - K_{Nahe}|$

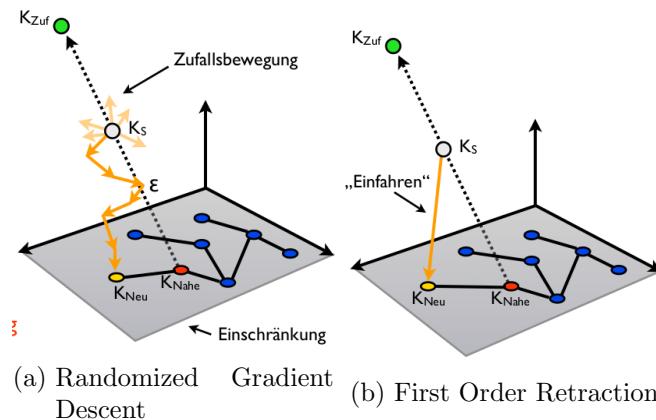


Abbildung 36: Methoden zur Berechnung der ConstrainedExtend-Funktion

Abbildung 37 stellt verschiedene Methoden der Stichprobenerzeugung gegenüber.
 Der RGD wird beispielsweise in IPoR II verwendet, wie in Abbildung 38 dargestellt.

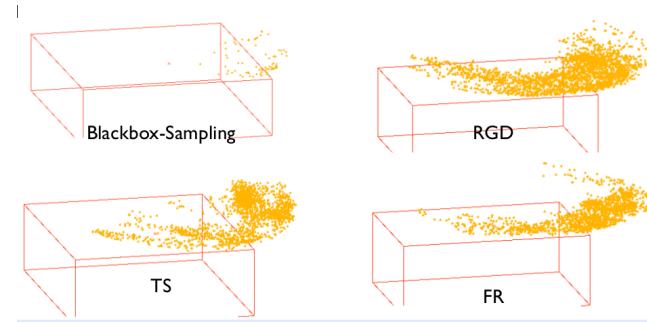


Abbildung 37: Vergleich der Stichprobenerzeugung: Jacobimatrix-basierte Ansätze bei komplexeren Einschränkungen im Vorteil

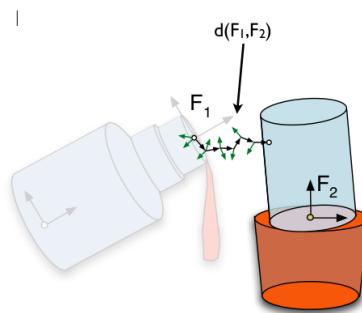


Abbildung 38: RGD mit Distanz $d(F_1, F_2)$

RRT – Erweiterungen:

- Connect-Heuristik:
 - Multiple Extend-Schritte in einer Iteration
- Bidirektional:
 - Wachsen eines RRTs in der Start- und Zielkonfiguration
 - Bäume wachsen aufeinander zu
 - Planungsproblem gelöst, wenn Bäume verbunden
 - Balanciert: gleiche Anzahl Knoten in beiden Bäumen
- dd-RRT [Jaillet05]:
 - Verwendung einer nicht-uniformen Stichprobenverteilung auf der Basis des aktuellen Suchbaums → Besseres Verhalten in eingeschränkten Regionen

3.7 Griffklassifikation

Bei IPoR II findet eine simple Abbildung auf Greifaktionen statt:

- Klassifikation des menschlichen Griffs im Segmentierungspunkt
- Abbildung auf vordefinierten Griff der gleichen Klasse für Roboterhand
- Bestimmung eines optimalen Griffs in Simulation
- Ausführung auf Roboter

Für die Klassifikation des menschlichen Griffs gibt es z.B. folgende Ansätze:

- Klassenhierarchie nach Cutkosky
- Training einer Support Vector Machine in jeder Hierarchieebene
- Hierarchische Auswertung der Support Vector Machines

Cutkosky-Hierarchie

Die Cutkosky-Hierarchie ist in Abbildung 39 dargestellt. Die in Abbildung 40 gezeigte hierarchische

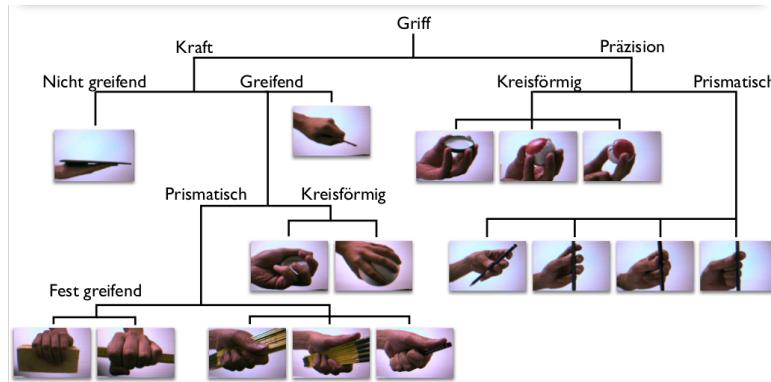


Abbildung 39: Die Cutkosky-Hierarchie

Netztopologie reflektiert Griffklassifikation nach Cutkosky. Jedes Netz wird mit entsprechenden Beispielen trainiert.

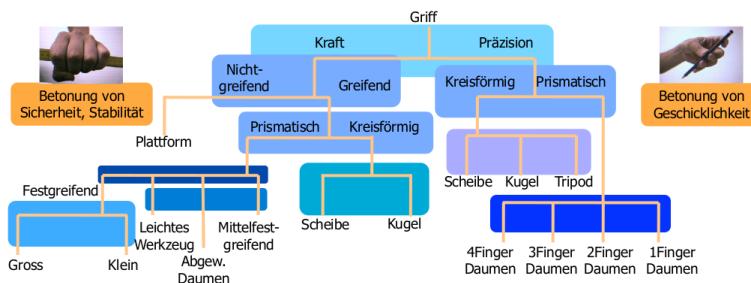


Abbildung 40: Hierarchische Netztopologie

3.8 Griffplanung

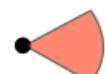
Zunächst einige relevante Definitionen:

- **Ziel:** Berechnung eines Griffes, d.h. Menge von Kontaktstellen zwischen Roboter und Objekt
- **Kontaktstellen**(→ Geometrisches Objektmodell, vgl. Abbildung 41):
 - Punktkontakt ohne Reibung (**A**)
 - Starrer Punktkontakt mit Reibung (**B**)
 - Nicht-starrer Punktkontakt mit Reibung („soft finger contact“) (**C**)
 - Flächenkontakte auf Basis von Punktkontakte
- **Wirkung auf Objekt:** Wrenchvektor: Kraft + Moment auf Objekt

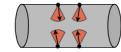
$$\begin{pmatrix} f \\ (c - s) \times f \end{pmatrix}$$

- Beschreibung der Griffqualität:

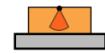
Cone Wrench Space (CWS): Kegel



Grasp Wrench Space (GWS): alle möglichen Summen aus jeweils einem Wrenchvektor jedes einzelnen Kegels



Task Wrench Space (TWS): aufgabenabhängig, hier: Knopf drücken



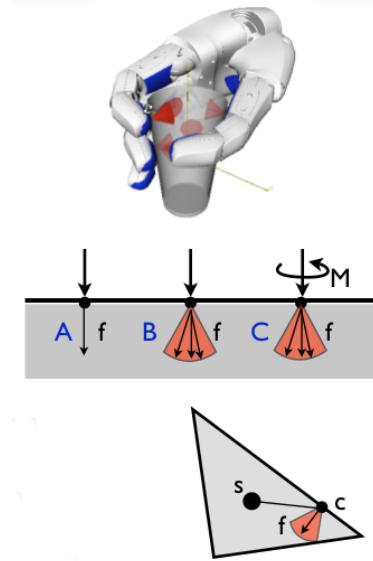


Abbildung 41: Griffplanung

- Eigenschaften eines Griffs: Widerstand gegen Stöße (beliebiger externer Wrench w):
 - Force-closure: $-w$ liegt im GWS
 - Form-closure: geometrische Einschränkung
- Qualitätsmaße für Griffe (Grasp quality measure)
 - Beispiel: größte Hyperkugel um 0, die im GWS liegt

Vorwärtsplaner: GrasPlt! Simulator für Greifbewegungen

Griffe werden bei der **Vorwärtsgriffplanung** in folgenden Schritten berechnet:

1. Setze Gelenkwinkel des Handmodells vor dem Zugreifen, z.B. prismatischer Kraftgriff
2. Setze 3d-Handmodell relativ zu Objekt vor dem Anrücken
3. Bewege Hand auf das Objekt zu
4. Schließe jeden einzelnen Finger bis auf Kontakt: Einfacher Algorithmus¹¹ (vgl. Abbildung 42)
 1. Schrittweise Änderung der Gelenkwinkel bis Hand komplett geschlossen
 2. Überprüfung der Kollisionen in jedem Schritt → geom. Objektmodell
 3. Bei Kollision: gebe vorherige Gelenkwinkel zurück
5. Bestimme Kraftkegel in allen Kontaktpunkten
6. Berechne Griffqualität
7. * Iteriere bis Griff mit hoher Griffqualität gefunden

¹¹ Spezialisierte Algorithmen basierend auf Distanz: C2A [Tang2009]

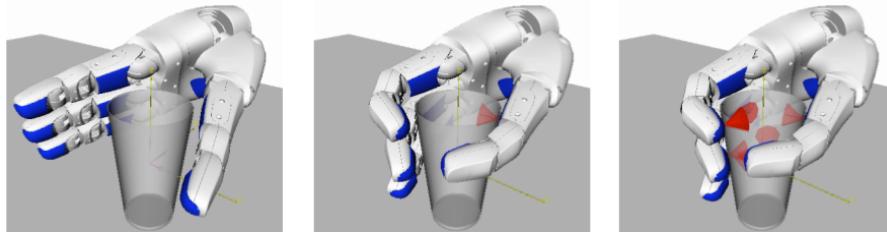


Abbildung 42: GraspIt!

- Beliebige Robotersysteme
- Beliebige Objekte, Hindernisse
- Verschiedene Qualitätsmaße für Griffe
- Soft finger contacts
- Physikengine

Ein vorwärtsgerichtetes Greifplanungsverfahren für die Barretthand¹² auf Basis des GraspIt! Simulators läuft folgendermaßen ab:

- Vorgabe einer Griffform („preshape“, hier zwei vordefinierte preshapes: zylindrisch und kugelförmig)
- Vorgabe einer Startpose und Anrückbewegung
- Durchführung des Griffs
- Evaluation mit Qualitätsmaß

Die Bestimmung der Greifstrategie (Preshape und Startlage) erfolgt auf Basis einer Zerlegung des zu greifenden Objekts in Objektprimitive (Kugeln, Zylinder, Kegel, Quader). Dabei existiert eine Vorgabe einer Menge von Greifstrategien für jedes der Objektprimitive. Die Anrückbewegung erfolgt dann linear in z -Richtung bis zu einem Kontakt, danach Backtracking. Abbildung 43 zeigt auf der linken Spalte generierte Greifstrategien für verschiedene Objekte und auf der rechten Seite deren Umsetzung.



Abbildung 43: Greifplanungsverfahren für die Barretthand

¹²[Miller03]: Automatic Grasp Planning Using Shape Primitives

Das **Lernen von Griffen durch PdV** ist im folgenden noch einmal zusammengefasst:

1. Beobachtung des Menschen
 - Mensch demonstriert Griffbewegung
 - Bestimmung der Griffform basierend auf Cutkosky-Hierarchie, z.B. sphärischer Präzisionsgriff
 - Bestimmung der Anrückbewegung auf das Objekt
2. (Vorwärtsgerichtete) Griffplanung zur Bestimmung von Kontaktstellen auf dem Objekt
 - Abbildung der menschlichen auf eine Roboter-Griffform
 - Abbildung der menschlichen auf eine Roboteranrückbewegung
 - Anrücken an Objekt und Schließen der Hand
 - Bestimmung der Kontaktstellen mit geometrischem Objektmodell
 - Berechnung des gewählten Qualitätsmaßes für Griffe
3. Ausführung auf dem Robotersystem: Griffkraft manuell nachjustiert

4 Aktionsplanungsverfahren

4.1 Definitionen

4.1.1 Planungsproblem

Unter einem Planungssystem für Roboter versteht man allgemein ein System, das ausgehend von einem Anfangszustand und der Beschreibung eines gewünschten Zielzustandes eine Folge von Aktionen generiert, die das betrachtete System von seinem Anfangszustand schrittweise in den gewünschten Zielzustand überführen.

4.1.2 Aktionsplanung

Gegeben sind Startzustand, Zielzustand und eine Menge möglicher Aktionen. Gesucht wird eine Sequenz/eine Menge von Aktionen, die den Startzustand in den Zielzustand transformieren.

Alternativen sind das Erreichen einer Menge von Zielen, das Erreichen von Zielen mit gegebenen Einschränkungen und das Erreichen von Zielen mit gegebenen Richtlinien.

Beispiel eines Planungsproblems – Verteilen von Post durch einen Roboter:

- Gegeben: Menge an Paketen mit jeweiligem Adressaten, Menge an Adressaten, Roboteraktionen: Greifen; Fahren; Ablegen
- Gesucht: Handlungsfolge, die alle Pakete an ihren Bestimmungsort bringt
- Einschränkungen: Batterieladung
- Richtlinien: Zeitoptimal, wegoptimal, unterschiedliche Prioritäten

Probleme:

- Teilweise unbekannte Umwelt
- Messunsicherheiten
- Effekte von Aktionen nicht immer deterministisch
- Externe Ereignisse können das Ergebnis beeinflussen
- Eigene Aktionen können das Ergebnis negativ beeinflussen

- Randbedingungen (Zeit, Ressourcen, etc.)
- Beachtung von Richtlinien

Ansatz:

- Explizite Repräsentation von Zustand, Zielen, Aktionen und Plänen
- Flexible Planungsalgorithmen und Suchstrategien (die Ordnung des Planungsproblems ist nicht notwendigerweise die Ordnung der Planausführung)
- Zerlegen des Ziels (Teilziele können mehr oder weniger unabhängig erreicht werden)

Vereinfachende Annahmen:

- Bekannter Ausgangszustand
- Deterministische Aktionen
- Keine externen Störungen
- Einfache Aktionsbeschreibung (keine bedingten, quantifizierten oder funktionalen Effekte)
- Aktionen nur sequentiell
- Keine Aktionen auf Sensorik (keine Verzweigungen im Plan)
- Keine Zeitbeschränkung
- Ausreichende Ressourcen

4.2 Suchverfahren

Das Planen kann als Suchproblem formuliert werden, indem Anfangszustand, Zielzustände und Zieltests, Nachfolge-Funktionen und ein optionales Gütekriterium für die Lösung definiert werden. Lösungen des Suchproblems stellen Pfade im Zustandsraum dar. Die verschiedenen Suchverfahren unterscheiden sich anhand ihrer Expansionsstrategien. Ein Beispiel ist in Abbildung 44 dargestellt. Bewertung von

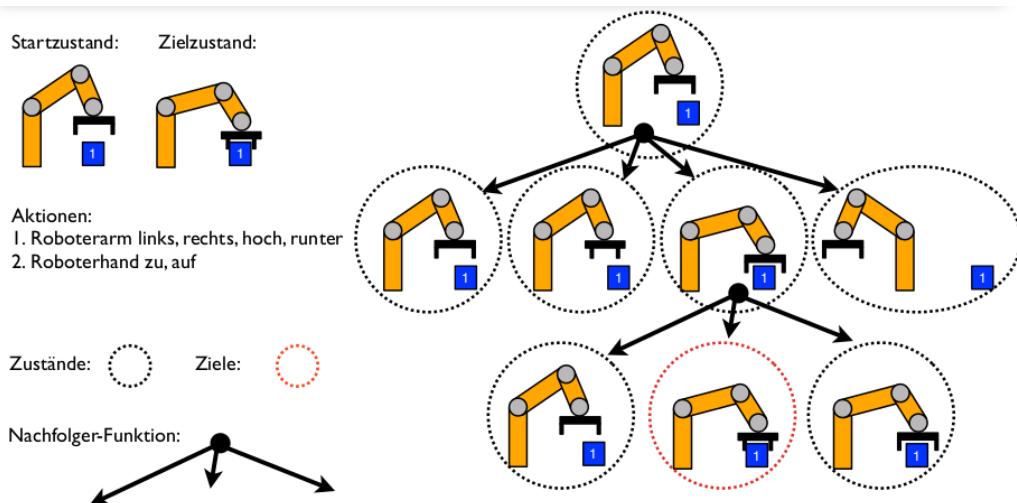


Abbildung 44: Beispiel für ein Suchverfahren

Suchverfahren:

- Vollständigkeit (wenn eine Lösung existiert, wird sie auch gefunden)
- Optimalität (wenn eine Lösung gefunden wird, ist diese auch optimal bezüglich eines Gütekriteriums)
- Zeit Komplexität (wie lange dauert es eine Lösung zu finden)
- Raum Komplexität (wie viel Speicher wird für die Suche benötigt)

4.2.1 Uninformierte Suche: Analytische Verfahren (Baumsuche)

Breitensuche

1. Beginn am Ausgangszustand
2. Prüfe alle Knoten gleicher Tiefe
3. Gehe dann eine Stufe tiefer

Die Breitensuche ist **vollständig**.

Tiefensuche

1. Beginn am Ausgangszustand
2. Suche bis zur maximalen Tiefe
3. Weiter im tiefsten Knoten, der noch nicht durchsuchte Kanten hat

Die Tiefensuche ist **nicht vollständig** falls Suchraum und Suchtiefe unbeschränkt sind.

Bidirektionale Suche

1. Beginne eine Breitensuche am Ausgangszustand und eine weitere am gewünschten Zielzustand
2. Halte, wenn bei beiden Suchen ein gleicher Zustand erreicht wurde
3. Die Lösung setzt sich aus den beiden Pfaden, mit denen dieser gleiche Zustand erreicht wurde, zusammen

Die Bidirektionale Suche ist **vollständig**.

Progression vs. Regression

Progression (Vorwärtssuche)

- Wähle Aktion, deren Vorbedingungen erfüllt sind
- Fortfahren, bis Zielzustand erreicht ist

- + Einfacher Algorithmus
- Suche kann sehr breit werden

Regression (Rückwärtssuche)

- Wähle Aktion, deren Effekt ein nicht erfülltes Teilziel erfüllt
- Füge nicht erfüllte Vorbedingungen der Aktion zu den Teilzielen hinzu
- Fortfahren, bis keine unerfüllten Teilziele mehr existieren

- + Fokus auf der Erfüllung von Teilzielen
- Regression ist unvollständig für funktionale Effekte

4.2.2 Informierte Suche: Heuristiken

Ist möglich, wenn zusätzliches Wissen zugänglich ist, z.B. Ignorieren irrelevanter Informationen und Ausschluss von Aktionen, die keine Annäherung zum Zielstand bringen.

A* hat als zusätzliches Wissen die Schätzung der ‚Distanz‘ zwischen einem Zwischenzustand und dem gewünschten Endzustand. A* ist eine Baumsuche mit nicht systematischer Suche entlang der Baumstruktur.

- Heuristik $h(n)$: Funktion, die die Distanz (Kosten) des Zustandes n zum Zielzustand schätzt
- Funktion $g(n)$: Ermittelt die tatsächlichen Kosten vom Ausgangszustand zum Zustand n
- Suche jeweils in dem Knoten fortsetzen, in dem $f(n) = g(n) + h(n)$ minimal ist.

A* ist **vollständig** und **optimal** falls $h(n)$ eine untere Schranke für die tatsächlichen Kosten darstellt, d.h. die tatsächlichen Kosten unterschätzt.

Dekomposition oder Zerlegung in unabhängige Teilprobleme. Das zusätzliche Wissen ist die vollständige Unabhängigkeit bestimmter Teilaufgaben.

Lineare Planung

Nichtlineare Planung

- | | |
|--|---|
| <ul style="list-style-type: none"> • Lösungen von Teilzielen werden nacheinander geplant • Stack noch offener Teilziele
<ul style="list-style-type: none"> + Einfache Suchverfahren + Effizient, wenn Teilziele unabhängig - Kann suboptimale Pläne erzeugen - Unvollständig | <ul style="list-style-type: none"> • Verschachteltes Lösen der Teilziele • Menge noch offener Teilziele
<ul style="list-style-type: none"> + Vollständig + Kürzere Pläne möglich - Größerer Suchraum |
|--|---|

Drei weitere Arten der informierten Suche/Heuristiken sind in Unterabschnitt 4.3, Unterabschnitt 4.4 und Unterabschnitt 4.5 ausgeführt.

4.3 Lineare Planung

- Jedes Teilziel wird für sich gelöst. Dann Lösung des nächsten Teilziels
- Teilziele auf einem Stack (feste Reihenfolge)
- Voraussetzung: Teilzeile sind (weitestgehend) unabhängig
- Lineare Planung ist effizient, wenn die Voraussetzung erfüllt ist

Klassisches Beispiel: **Blockwelt**

Szenario: Tisch mit Blöcken, Greifer
Operatoren:

- $Pickup(x)$: Block x vom Tisch greifen Nach der Anwendung von $Putdown(2, s_0)$ gilt $OnTable(2, s_1)$.
- $Putdown(x)$: Block x auf Tisch legen
- $Stack(x, y)$: x auf y ablegen
- $Unstack(x, y)$: x von y aufnehmen

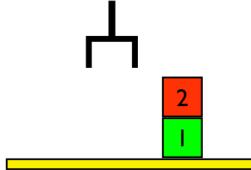
4.3.2 Formalisierung: Situationskalkül

Das Situationskalkül benutzt ausschließlich die Prädikatenlogik 1. Ordnung zur Planung. Sämtliche Annahmen, die von der Welt gemacht werden, müssen daher durch Axiome formuliert werden. Das Szenenmodell ist somit eine Konjunktion von Prädikaten in Prädikatenlogik.

Beispiel einer Situation zum Zeitpunkt s_0 :

$$\text{Ontable}(1, s_0) \wedge \text{On}(2, 1, s_0) \wedge \text{Holding}(\text{NIL}, s_0) \wedge \text{Clear}(2, s_0)$$

Gesucht: Situation mit $\exists t : [\text{Holding}(2, t)]$



- **Aktionen:** werden durch Funktionen dargestellt (Bsp. Blockwelt $\text{Stack}(x, y, s_t)$).
- Situationen: sind Zustände der Umwelt. Sie werden durch die Anwendung von Aktionen auf den Anfangszustand dargestellt
- Situationsabhängige Attribute: Die Attribute selbst werden durch Funktionen oder Variablen dargestellt. Ihre Auswertung wird durch das Prädikat $\text{holds}(\text{fluent}, \text{situation})$ vorgenommen.
- Situationsunabhängige Attribute: können einfach als Prädikate formuliert werden.
- **Vorbedingungen** von Aktionen (Bsp. Blockwelt $\text{Holding}(x, s_t) \wedge \text{Clear}(y, s_t)$)
- **Nachbedingungen** von Aktionen (Bsp. Blockwelt $\neg \text{Holding}(x, s_{t+1}) \wedge \text{Holding}(\text{NIL}, s_{t+1}) \wedge \neg \text{Clear}(y, s_{t+1}) \wedge \text{On}(x, y, s_{t+1}) \wedge \text{Clear}(x, s_{t+1})$)
- Ergebnisse von Aktionen: Konsequenz einer Aktion auf die folgende Situation.

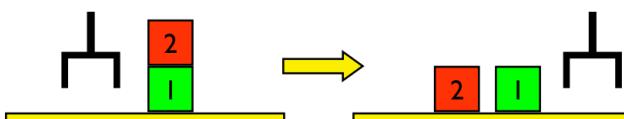
Beispiel – Lineare Planung in der Blockwelt:

$$\text{Initial (0): } \text{Ontable}(1, s_0) \wedge \text{On}(2, 1, s_0) \wedge \text{Holding}(\text{NIL}, s_0) \wedge \text{Clear}(2, s_0)$$

$$\text{Ziel (t): } \text{Ontable}(1, s_t) \wedge \text{Ontable}(2, s_t) \wedge \text{Holding}(\text{NIL}, s_t) \wedge \text{Clear}(1, s_t) \wedge \text{Clear}(2, s_t)$$

Planer:

1. Ziel ($\text{Ontable}(2, s_t)$): $\text{Unstack}(2, 1, s_0); \text{Putdown}(2, s_1);$
2. Ziel ($\text{Clear}(1, s_t)$): Wird per Zufall mit 1. Ziel abgedeckt



Formalisierung der Blockwelt-Operatoren – Rahmenproblem (Frame Problem):

- Operator $\text{Putdown}(x, s_t)$:
 - Vorbedingung: $\text{Holding}(x, s_t)$
 - Nachbedingung: $\text{Ontable}(x, s_{t+1}) \wedge \text{Clear}(x, s_{t+1}) \wedge \text{Holding}(\text{NIL}, s_{t+1}) \wedge \neg \text{Holding}(x, s_{t+1})$
- Beispiel: Zustand s_0 : $\text{Holding}(2, s_0) \wedge \text{Ontable}(1, s_0)$
- Nach Anwendung von $\text{Putdown}(2, s_0)$ gilt $\text{Ontable}(2, s_1)$
- Intuitiv ist klar: Es gilt auch $\text{Ontable}(1, s_1)$. Dies kann aber nicht formal aus $\text{Ontable}(1, s_0)$ abgeleitet werden.

- Also: $Ontable(2, s_1)$ und $Ontable(1, s_1)$ kann nicht abgeleitet werden

⇒ **Rahmenaxiome:** Es muss nicht nur formuliert werden, was ein Operator bewirkt, sondern auch was er *nicht* verändert. Die Nachbedingungen müssen also alle möglichen Nicht-Veränderungen (Rahmenaxiome) enthalten. Das Modellieren wird dadurch sehr lästig und das Planen darauf sehr komplex

→ STRIPS

4.3.3 STRIPS

= STanford Research Institute Problem Solver. Ein linearer Planer. Wie beim Situationskalkül haben Operatoren Vor- und Nachbedingungen. Die Nachbedingungen bestehen jedoch aus einer **Add-** und einer **Delete-Liste**. Die Add-Liste enthält diejenigen Attribute, die dem neuen Zustand durch das Ausführen der Aktion hinzugefügt werden. Die Delete-Liste enthält diejenigen Attribute, die aus dem neuen Zustand durch das Ausführen der Aktion negiert werden.

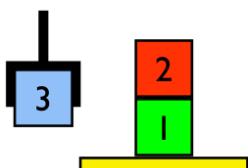
Annahme: Ein Operator ändert, wenn man ihn in der Welt ausführt, genau das, was in seiner Nachbedingung angegeben ist (d.h. Attribute, die nicht in der Definition des Operators erwähnt werden bleiben unverändert).

Operator

- Operator: o
- Aktion: $Pickup(x)$
- Vorbedingung: $Holding(NIL) \wedge Clear(x) \wedge Ontable(x)$
- Nachbedingungen:
 - Add-Liste $add(o)$: $Holding(x)$
 - Delete-Liste $del(o)$: $Holding(NIL), Clear(x), Ontable(x)$

Planung

- Situation: Ansammlung von Fakten ($\{Ontable(1), On(2, 1), Holding(3), \dots\}$)



- Planung: Suche eines Weges vom Initialzustand zum Zielzustand
- Ziel: Nicht ein Zielknoten, sondern eine Menge von Knoten (Ziel gibt nicht immer alles vor, bspw. sagt $On(3, 2)$ nichts über Block 1 aus).

→ Rückwärtssuche von der Menge der Zielknoten (**zielzentriert**) mit Mittel-Ziel Analyse mit Zielkeller (**MZAMZK**):

- Eingabe: Menge der Ziele Z und Startsituation S
- Global: Menge der Operatoren
- Variable: Aktuelle Situation S_t
- Ausgabe: Plan P (lineare Ordnung von Operatoren)

Algorithmus (MZAMKZK) – Wiederhole solange offene Ziele vorhanden sind:

1. Wähle das erste Ziel g aus (top of stack)
2. Wähle einen anwendbaren Operator o , dessen Add-Liste g enthält
3. Falls kein solcher Operator existiert dann Backtracking und weiter mit 1.
4. Andernfalls füge alle nicht erfüllten Vorbedingungen von o zu den Zielen hinzu (push on stack)

MZAMZK kann um Kritiker erweitert werden:

- Einführung eines Kritikers zum Plan-Debugging
- Finden von sich aufhebenden Operationen
- Löschen der entsprechenden Planteile ($Pickup(1)$ direkt vor $Putdown(1)$ wird gelöscht)

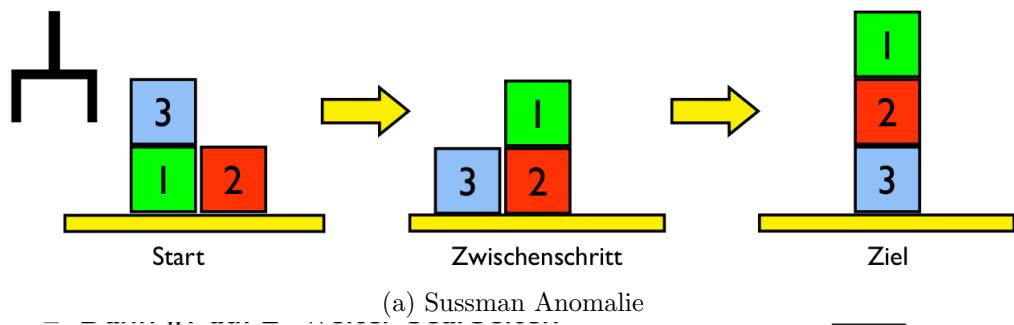
Teilzielinteraktionen:

- Negative TZI: Bearbeitung eines Teilziels zerstört Teile der Bearbeitung eines anderen Ziels
 - Positive TZI: Bearbeitung eines Teilziels erledigt Teile der Bearbeitung eines anderen Ziels mit
- STRIPS berücksichtigt positive TZI, aber keine negativen TZI.

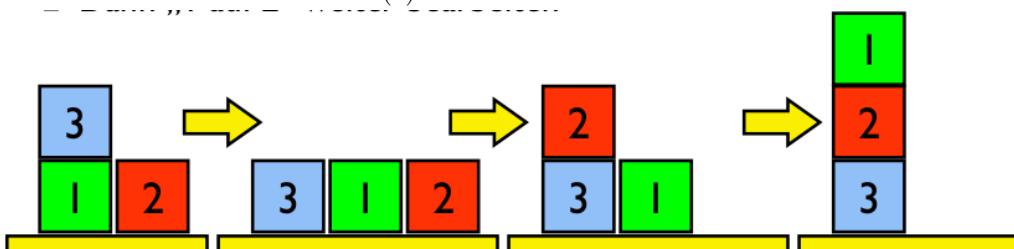
Sussman Anomalie (suboptimale Pläne)

- Ziel $On(1, 2) \wedge On(2, 3)$ (vgl. Abbildung 45a)
- Erzeugter Plan: $Unstack(3, 1)$, $Putdown(3)$, $Pickup(1)$, $Stack(1, 2)$, $Unstack(1, 2)$, $Putdown(1)$, $Pickup(2)$, $Stack(2, 3)$, $Pickup(1)$, $Stack(1, 2)$
- Die Sussman Anomalie kann mit POP (Unterabschnitt 4.4) oder Einführung eines Kritikers gelöst werden.
- $Stack(1, 2); Unstack(1, 2)$ hebt sich auf, wird also gelöscht; nun sind $Pickup(1); Putdown(1)$ direkt hintereinander, werden also auch gelöscht

→ erhalte optimierten Plan: $Unstack(3, 1); Putdown(3); Pickup(2); Stack(2, 3); Pickup(1); Stack(1, 2)$



(a) Sussman Anomalie



(b) Lösung der Sussman Anomalie: Verzahnung von Teilzielen bei der linearen Planung nicht möglich

Abbildung 45

One way Rocket- und Zuweisungs-Problem (unlösbarer Probleme):

- Geg.: zwei Objekte o_1 und o_2 , Flugzeug p und Flughafen a_1
- Aktionen: $load(a, o, p)$ und $unload(a, o, p)$
- Ziel: $at(a_2, o_1)$, $at(a_2, o_2)$
- definiere: $fly(p, a_1, a_2)$ mit
 - Vorbedingung: $plane(p) \wedge at(p, a_1) \wedge haveFuel(p)$
 - $add(fly)$: $at(p, a_2)$
 - $del(fly)$: $at(p, a_1), haveFuel(p)$

→ kann mit linearer Planung nicht gelöst werden, da immer ein Teilziel vollständig erfüllt wird und damit die Vorbedingung für das zweite Ziel nicht mehr gegeben ist

4.3.4 Diskussion

Vorteile:

- Reduzierter Suchraum, weil Teilziele nacheinander gelöst werden
- Vorteile, wenn Teilziele unabhängig
- Produziert nur gültige Pläne

Nachteile:

- Kann suboptimale Pläne erzeugen
- ist unvollständig

4.4 Nichtlineare Planung

Planung als Suche mit einer (Teil-)Ziel-Menge (statt Stack). Arbeitet mit einer Menge von Zielen. Im Suchraum werden alle möglichen Teilziel-Anordnungen dargestellt: Teilzielinteraktionen durch Ziel-Verzahnung bei der Planung betrachtet. Abhängigkeiten der einzelnen Operatoren können geplant werden. Plan kann z.B. in Form eines Vorranggraphen gespeichert werden. Nichtlineare Planungssysteme (NLP) folgen einer **Least Commitment Strategie** (Prinzip der geringsten Festlegung):

- Es gibt keine a priori Festlegung der Reihenfolge, in der die Ziele erreicht werden sollen
- Die Reihenfolge, in der Aktionen ausgeführt werden, wird nur soweit nötig festgelegt
- Variablen werden nur falls notwendig instanziert

Vorteile:

- Erzeugt nur gültige Pläne
- vollständig
- Optimalität bzg. irgendeines Kriteriums kann erreicht werden

Nachteile:

- Größerer Suchraum, alle Teilzielreihenfolgen müssen beachtet werden
- Komplexere Algorithmen

Ein **nicht-linearer Plan** P ist eine Datenstruktur mit folgenden Komponenten:

- eine Menge von Planschritten S mit Operation o ($S : o$)
- eine Menge von Ordnungsbedingungen, die die Reihenfolge von Planschritten angeben ($S_i < S_j$), d.h. „ S_i vor S_j “
- eine Menge von Variablenbedingungen ($x = t$), wobei x eine Variable und t ein Term ist, und Ungleichungsbeschränkungen
- eine Menge von kausalen Beziehungen (causal links), die Zusammenhänge zwischen Planschritten beschreiben, d.h. S_i erzeugt die Vorbedingung c für S_j
- Menge der offenen Vorbedingungen

4.4.1 Einfacher Algorithmus

Wiederhole solange Zielmenge G nicht leer ist

- Wähle g aus G
- Wenn g nicht im Zustand enthalten ist
 - Wähle Operator o , dessen Add-Liste g enthält
 - Füge o zu den auszuführenden Operationen hinzu
 - Füge Vorbedingungen von o zu G hinzu

4.4.2 Partial Order Planning (POP)

Abhängigkeiten der einzelnen Operatoren können geplant werden und der Plan kann z.B. in Form eines Vorranggraphen gespeichert werden (Beispiel in Abbildung 46)

POP basiert auf einer Suche im Raum der möglichen Pläne, nicht auf einer direkten Suche im Zu-

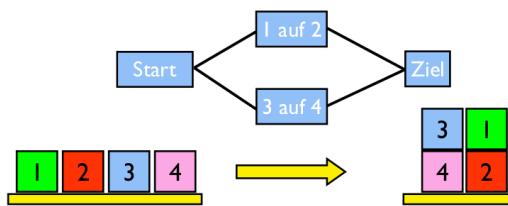


Abbildung 46: Vorranggraph

standsraum.

- Knoten: (unvollständige) Pläne
- Kanten: Planmodifikationsschritte
 - Einfügen von Planschritten
 - Anordnen von Planschritten
 - Instanziieren von Variablen

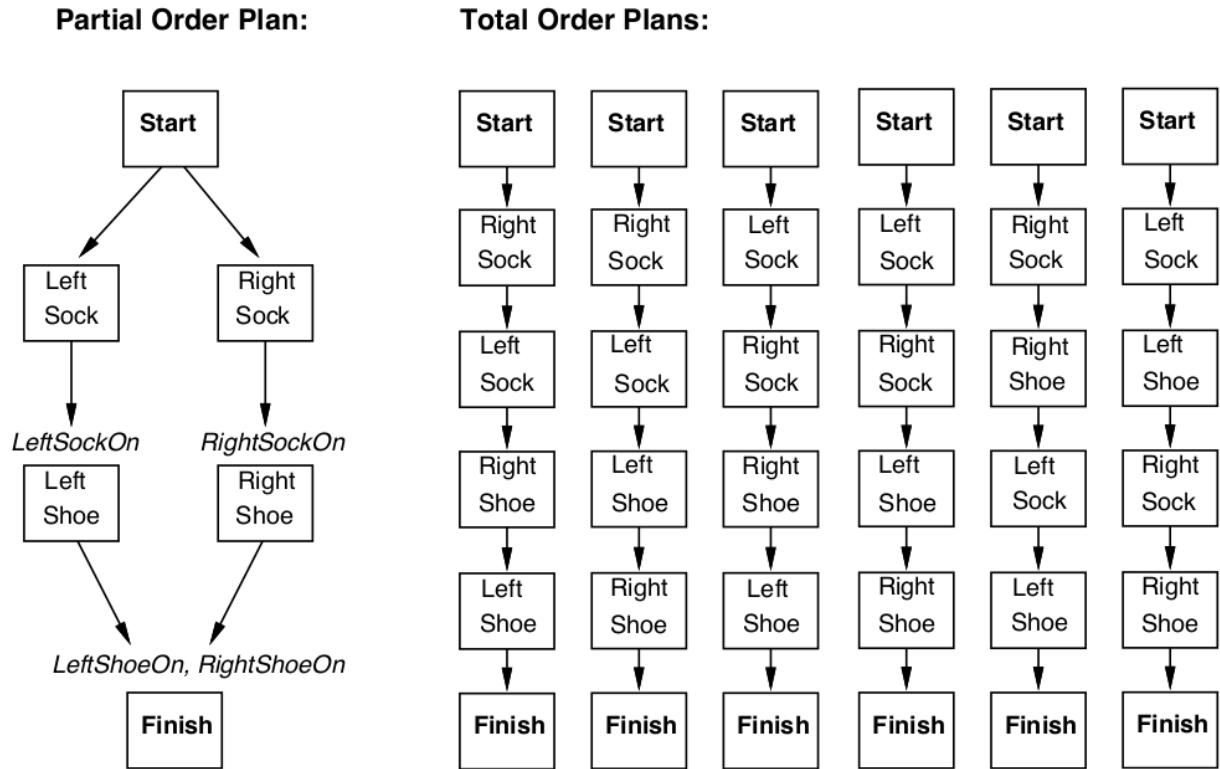


Abbildung 47: Partial vs. Total Order Planning (bei linearer Planung)

POP ist **vollständig** und erzeugt konsistente nicht-lineare Pläne.

Jede Linearisierung eines solchen Plans ist eine Lösung des Planungsproblems.

Bei NLP POP kann es zu Konflikten kommen, wie in Abbildung 48 dargestellt. Indem man **Kritiker**

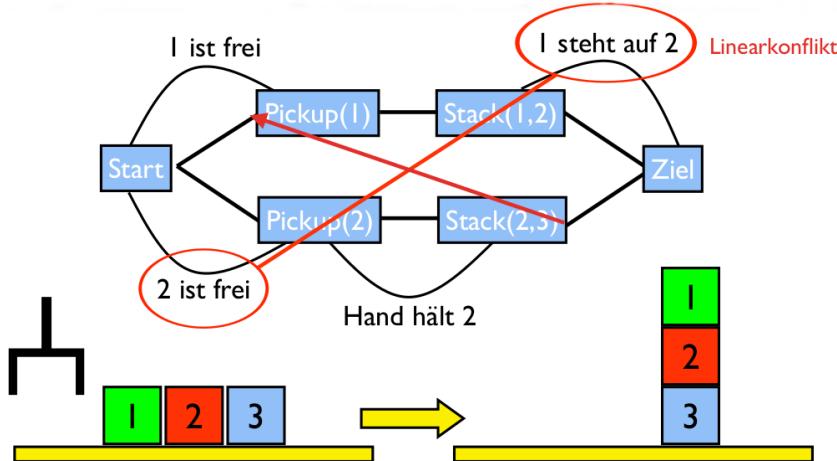


Abbildung 48: POP Konflikte

in NLP einführt kann man nach der Konfliktauflösung (Abbildung 48) den Plan weiter optimieren, bspw.

- Redundante Kanten entfernen
- Operatoren löschen, von denen keine Abhängigkeiten ausgehen
- Hilfreiche Interaktionen entdecken
- Umwege entfernen (siehe Sussman-Anomalie)

4.4.3 Total vs. Partial Order Planning

Total Order

- Plan entspricht immer einer strikten Sequenz von Aktionen

+ Einfachere Algorithmen

- Keine nebenläufigen Pläne

- Enthält evtl. unnötige Abhängigkeiten

Partial Order

- Aktionen können ungeordnet sein
- Nur notwendige Abhängigkeiten werden geplant
- Plan muss evtl. vor der Ausführung linearisiert werden

+ Enthält nur notwendige Abhängigkeiten

+ Nebenläufige Pläne darstellbar

- Feststellung der erfüllten Teilziele zu einem Zeitpunkt schwierig

- Komplexere Darstellung

4.5 Hierarchische Planung

Hierarchische Aufgabennetzwerke (Hierarchical Task Networks, HTN), Planung in Dimension der Abstraktion.

- Komplexe Pläne haben meistens erkennbare Strukturen
- Diese Struktur kann oft in Form von Hierarchien ausgedrückt werden
- Teilpläne sind oft voneinander unabhängig
- Hierarchische Planung versucht die Dimensionalität des Problems zu reduzieren, im Gegensatz zur vollständigen Suche im Plan- oder Zustandsraum

Klassische Planung kombiniert elementare Operationen, hierarchische Planung „entfaltet“ abstrakte Operationen (vgl. Abbildung 49). Beispiel – HTN:

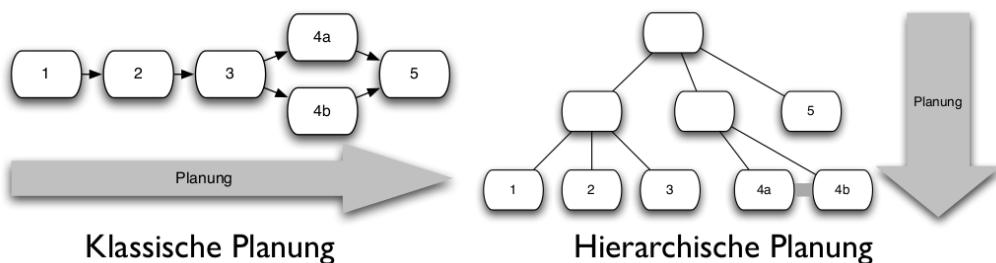


Abbildung 49: Unterschiedliche Planungsrichtungen für klassische Planung und hierarchische Planung

- Um zur Konferenz nach X zu kommen, muss man zum Flughafen kommen, ein Flugzeug nach X nehmen, zum Konferenzhotel fahren.
- Um zum Flughafen zu kommen, kann man entweder einen Bus nehmen oder ein Taxi.

- Wenn man genug Geld hat: Um ein Taxi nach Y zu nehmen entweder vorher anrufen oder ein Taxi herbeiwinken, einsteigen, sagen „Ich möchte nach Y “, warten bis Y erreicht ist, bezahlen, aussteigen.

Ein **abstrakter Plan** enthält zusammengesetzte Operatoren:

- Vorbedingungen, Effekte
- Methoden zur Zerlegung in Teilpläne
 - Aufbau der Teilplanstruktur
 - Parameter
 - Ähnliche Funktionsaufrufe

Ein voll instantiiertter Plan besteht nur aus primitiven Operatoren:

- STRIPS-ähnliche Beschreibung
- keine Vorbedingungen

4.5.1 HTNs – Suchraum

- Ziel ist eine abstrakte Aufgabe (Task), nicht ein Weltzustand (State)
- Operatoren im Suchraum
 - Zerlegung in Teilpläne
 - Parametrisierung
 - Konfliktlösung
- Algorithmus HTN-Planung:
 - Starte mit initialer abstrakter Aufgabe (nicht Weltzustand/Zielliste!!)
 - Baue das Aufgabennetzwerk durch wiederholtes Expandieren in Teilpläne auf, bis der Plan voll instantiiert ist
 - Expandieren durch Verwendung von Methoden, deren Anwendbarkeit gegeben ist

4.5.2 Simple Hierarchical Order Planner (SHOP)

SHOP Algorithmus

- Vorwärtssuche, lineare Planungsverfahren
 - Planung in Ausführungsreihenfolge
 - Tiefensuche
- Elementaroperatoren haben keine Vorbedingungen
- Keine parallelen Aktionen
- Sehr mächtige Operatorrepräsentation
- Effizienter Planungsalgorithmus
- Korrekt und vollständig

(:operator (!putdown ?block) ((holding ?block)) ((ontable ?block) (handempty)))	Delete Liste Add Liste
(:method (make-clear ?y) ((clear ?y)) (nil))	Vorbedingung Task-Liste
(:method (make-clear ?y) ((on ?x ?y)) ((make-clear ?x) (!unstack ?x ?y) (!putdown ?x)))	Vorbedingung Task-Liste

Abbildung 50: SHOP Beispieldomäne

Stärken des Algorithmus

- Methoden codieren Domänenwissen
- Methoden beinhalten Problemlösungswissen
- Abstraktion kapselt Operatorinteraktionen

Nachteile

- Immer noch NP-vollständig
- Terminiert nicht zwangsläufig (rekursives Anwenden von Methoden, Endlosschleifen sind evtl. schwer zu detektieren!)
- Plan kann erst im voll expandierten Zustand bewertet werden → Konflikte können evtl. erst dann entdeckt werden

Mögliche Erweiterungen

- Erkennung/Lösung von Konflikten
 - Teilzielinteraktion
 - Mehrfachverwendung/Wiederverwendung von Operatoren
- Methoden beinhalten Vorbedingungen und Effekte
 - Konflikte in der Planungsphase erkennen
- Methoden beinhalten Ressourcenangaben
 - Scheduling

5 Probabilistisches Entscheiden

Fundamentale Fragen:

- Was ist Entscheiden?
- Welche Arten von Systemen fällen Entscheidungen?
- Welche ist der allgemeinste Rahmen für eine Definition von Entscheidungssystemen?

5.1 Der rationale Agent

- Allgemeine Definition eines in eine Umwelt eingebetteten, handelnden Systems
- Beliebige Art von Umwelt
- Definiert durch den Agentenzyklus: **Perzeption – Entscheidung – Handlung** (Abbildung 51)
- Der Agentenzyklus ist eine andere Darstellung eines Regelkreises
- Entscheidung ist eine abstrakte Form der Regelung

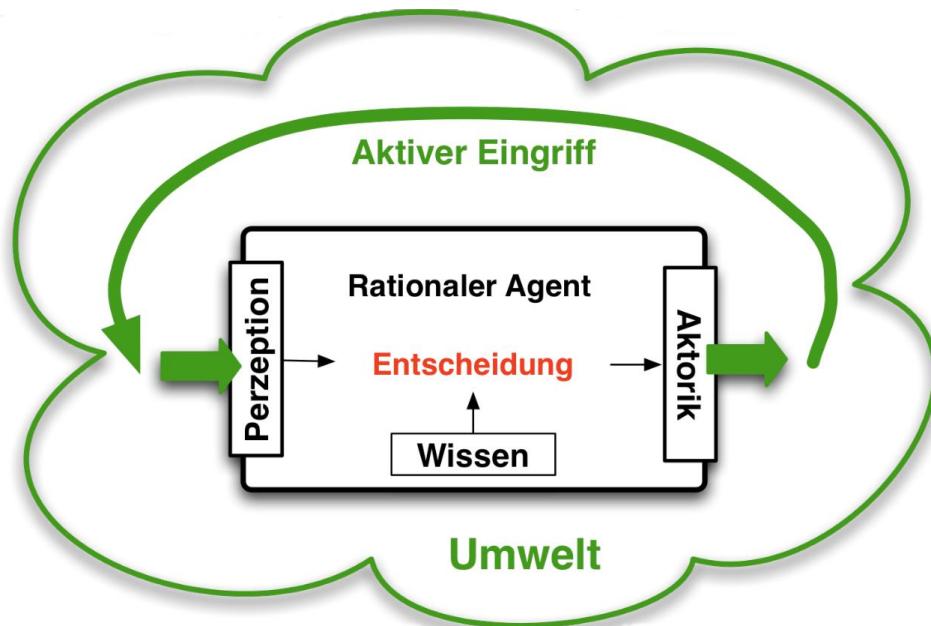


Abbildung 51: Der Agentenzyklus: Jede Modellbildung der Welt ist starke Reduktion

5.1.1 Beispiele

Schachcomputer

- Keine physische Umwelt
- Sehr beschränkte Umwelt
- Nur ein Ziel: das Spiel zu gewinnen
- Sensorik: Das Spielfeld wird komplett wahrgenommen
- Aktorik: Durchführung eines symbolischen Zugs
- Umwelt: semi-statisch, episodisch, diskret, deterministisch, vollständig beobachtbar, Multiagent

Industrieroboter

- Oftmals, bei Abwesenheit von Sensorik, kein Agent, sondern nur ein Werkzeug
- Physische Umwelt
- Beschränkte Umwelt
- Wenn überhaupt rationaler Agent, üblicherweise extrem begrenzter Entscheidungsrahmen
- Ziel: Erfolgreiche Ausführung einer Operation
- Umwelt: dynamisch, episodisch, diskret oder kontinuierlich, deterministisch, vollständig beobachtbar, Einzelagent oder Multiagent

Mensch

- Extrem komplexe Umwelt
- Sehr mächtige Perzeption
- Sehr wissensbasiert
- Vielschichtige Motivationen und Zielsetzungen
- Umwelt: dynamisch, sequentiell, kontinuierlich, stochastisch, unvollständig beobachtbar, Multi-agent

Serviceroboter

- Anforderungen dem Menschen viel ähnlicher als einem Industrieroboter
- Sehr komplexe Umwelt
- Vielschichtige Zielsetzung
- Es ist bei Servicerobotern ein ganz anderes Vorgehen, als bei Industrierobotern erforderlich
- Umwelt: wie beim Menschen

5.1.2 Umwelt

Für die verschiedenen Agenten werden sehr unterschiedliche Umgebungen benötigt. Ein Rahmen für eine nützliche und eindeutige Charakterisierung von Umgebungen ist nötig.

Anforderungen

- Die Charakterisierung muss fundamentale, nicht oberflächliche Eigenschaften aufzeigen
- Es muss ein Rahmen gebildet werden, in den verschiedenen Verfahren und Paradigmen eingeordnet werden können
- eine Leitlinie für die Entwicklung neuer Verfahren muss gebildet werden

Umwelten verschiedener Agenten können gut durch einige fundamentale, duale Eigenschaften unterschieden werden.

Eigenschaften

statisch	vs.	dynamisch
Der Agent ist das einzige Element, welches den Zustand der Umwelt verändert		Die Umwelt kann sich auch ohne Zutun des Agenten verändern
episodisch	vs.	sequentiell
Der zeitliche Verlauf des Geschehens ist in abgeschlossene Einheiten unterteilt, zwischen denen keinerlei Kausalzusammenhänge bestehen		Die vollständige zeitliche Vergangenheit hat Auswirkungen auf die Gegenwart, die Gegenwart auf die gesamte Zukunft
diskret	vs.	kontinuierlich
Die Zustände der Umwelt sind diskret, ebenso der zeitliche Verlauf des Geschehens		Der Zustandsraum der Umwelt ist kontinuierlich und die Zeit fließt ebenfalls kontinuierlich
deterministisch	vs.	stochastisch
Der Ausgang einer jeden Handlung ist eindeutig bestimmt		Eine Handlung kann mit bestimmten Wahrscheinlichkeiten zu verschiedenen Ausgängen führen
vollständig beobachtbar	vs.	unvollständig beobachtbar
Der Agent kann die komplette Umwelt jederzeit vollständig und exakt wahrnehmen		Der Agent kann die Umwelt nur eingeschränkt und fehlerbehaftet wahrnehmen
Einzelagent	vs.	Multiagent
Nur eine als Agent modellierbare Entität agiert in der Umwelt		Viele als Agenten modellierbare Entitäten agieren in der Umwelt und können kooperieren oder konkurrieren

Die Beschreibung der Umwelt durch das jeweils komplexere Merkmal wird nur gewählt, wenn das einfachere Merkmal im Szenario des Agenten keine hinreichende Beschreibung ist.

Hinreichend ist hierbei auch in dem Sinne zu verstehen, dass es oftmals noch keine ausreichenden Verfahren gibt, um die komplexeren Merkmale zu berücksichtigen.

5.1.3 Utility – Motivation des Agenten

- Ein Agent benötigt Motivation und Absichten als Fundament für Entscheidungen
- In der Entscheidungstheorie durch das Konzept der **Utility** abgebildet (Ursprung des Begriffs in der Ökonomie)
- Dieses Konzept ist deutlich allgemeiner als spezielle Ziele z.B. in der logikbasierten Planung

Konzept der Utility

- Die utility (auch: *utility function*) modelliert in der Ökonomie ein numerisches Maß der Befriedigung eines Agenten (Menschen) durch einen bestimmten Konsum
- In der Robotik/KI wird jedoch ein Motivationsmaß eines Agenten nicht *modelliert*, sondern *konstruiert*
- Die utility wird hier genutzt, um einem künstlichen Agenten bestimmte Motivationen einzupflanzen
- Die utility ist ein allgemeines Konzept: es können konkurrierende und ergänzende Zielsetzungen fusioniert werden
- Im Rahmen der utility können auch Absichten gegeneinander abgewogen werden

5.1.4 Problem

Die ursprüngliche, klassische Aktionsplanung nimmt an, dass die Umwelt statisch, episodisch, diskret, deterministisch und vollständig beobachtbar ist und zudem konkrete, nicht überlappende Ziele verfolgt. Für eine dynamische, sequentielle und kontinuierliche Umwelt gibt es Erweiterungen des klassischen Aktionsplanens, im Hinblick auf **stochastisch**, **vollständig beobachtbar** und **unterschiedlich stark konkurrierende Ziele** jedoch nicht.

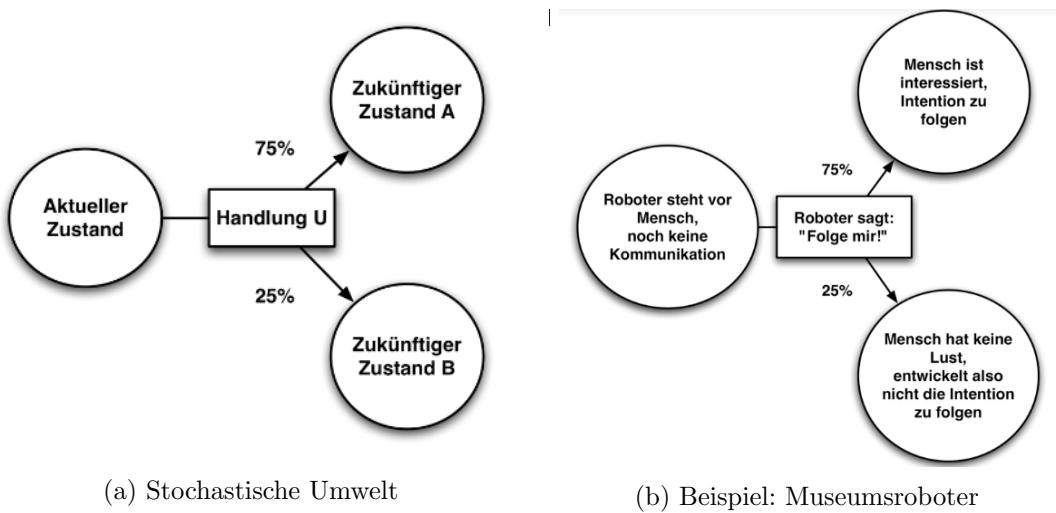


Abbildung 52

Abbildung 52a zeigt eine stochastische Umwelt. Übergangswahrscheinlichkeiten beschreiben hierbei den Ausgang von Handlungen. In stochastischen Umgebungen ist es daher essenziell, dass mit numerischen Wahrscheinlichkeiten umgegangen werden kann, jedoch ist das in der klassischen, logikbasierten Aktionsplanung nicht der Fall, weshalb **probabilistische Methoden** (wie MDPs) zum Einsatz kommen.

5.2 Markov Prozesse

- ein verbreitetes Werkzeug für die Modellierung stochastischer Prozesse
- zentrale Eigenschaft: die zukünftige Entwicklung kann durch Kenntnis einer **begrenzten** Vorgeschichte prognostiziert werden
- erst diese Markoveigenschaft erlaubt die Prognose bezüglich der Zukunft in einem solchen stochastischen Prozess
- es gibt zwei Arten von Markov Prozessen: zeitdiskrete und zeitstetige Markov Prozesse

Zeitdiskreter Markov Prozess:

- werden auch Ketten genannt
- Übergänge in einer Markov Kette sind Sprünge in einer diskreten Zeit
- Mathematisch wird die Wahrscheinlichkeit für einen zukünftigen Zustand als bedingte Wahrscheinlichkeit, abhängig von einer endlichen Zahl an vergangenen Zuständen modelliert
- die Ordnung einer Markov Kette gibt an, welcher Ausschnitt der Vergangenheit eine Rolle spielt
- die Ordnung ist immer endlich, sonst wäre die Markov Eigenschaft nicht erfüllt
- je geringer die Ordnung, desto einfacher die Modellierung

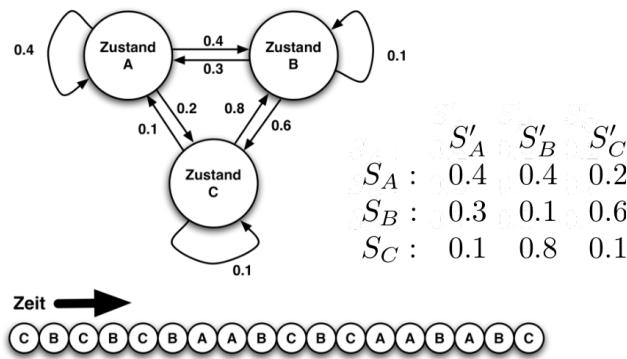


Abbildung 53: Beispiel – Markovkette 1. Ordnung

Markovkette 1. Ordnung (Beispiel in Abbildung 53)

- nur die Gegenwart spielt eine Rolle für die Betrachtung der Zukunft
- die Wahrscheinlichkeit des Nachfolgezustands hängt nur vom aktuellen Zustand ab
- $Pr(X_{n+1} = x | X_n = x_n, \dots, X_1 = x_1, X_0 = x_0) = Pr(X_{n+1} = x | X_n = x_n)$

Markovketten: Modellierung der Welt

- Markovketten können von Agenten zur Modellierung von stochastischen Umgebungen genutzt werden
- Bei diskreten Markovketten muss die Welt dazu in eine diskrete Zustandsmenge zerlegt werden
- Die Zerlegung kann für verschiedene Umweltaspekte getrennt erfolgen, z.B. Roboterposition, Objektform oder Intention eines Menschen
- Welt als diskrete Zustände – einfaches Beispiel: Unterteilung von Räumen in diskrete Regionen (die Position eines mobilen Roboters wird durch die Region, in der er sich befindet, repräsentiert)
- Welt als stochastischer Prozess: Bestimmte Auslöser führen zu stochastischen Zustandsübergängen (Bayesianische Modellierung), z.B. Handlung eines Agenten:
Beispiel Museumsroboter:
 - Der Roboter steht vor einem Mensch, es fand noch keine Kommunikation statt
 - Möglichkeit 1: Roboter sagt: Folge mir!; damit ergibt sich eine Wahrscheinlichkeit von 75%, dass der Mensch interessiert ist und die Intention hat zu folgen und 25%, dass der Mensch keine Lust hat, also nicht die Intention entwickelt zu folgen
 - Möglichkeit 2: Roboter sagt: Ich habe etwas interessantes zu zeigen. Folge mir bitte!; damit steigt die Wahrscheinlichkeit, dass der Mensch folgen will auf 90%, wohingegen er mit 10%iger Wahrscheinlichkeit nicht folgen will
 - d.h. durch die Wahl der Handlung kann der Agent die **Übergangswahrscheinlichkeiten steuern!!!** (Siehe auch Abbildung 54).

5.2.1 Markov Entscheidungsprozesse

Ketten von Handlungen bzw. Auslösern führen zu Bayes-Netzwerken. Für die Entscheidungsfinder in diesen stochastischen Prozessen existiert ein fundiertes Modellierungsrahmenwerk spezieller Bayes-Netze: **Markov Decision Processes** (MDPs). Hierbei kann der Agent, im Unterschied zu Markov-Ketten, aktiv Entscheidungen treffen.

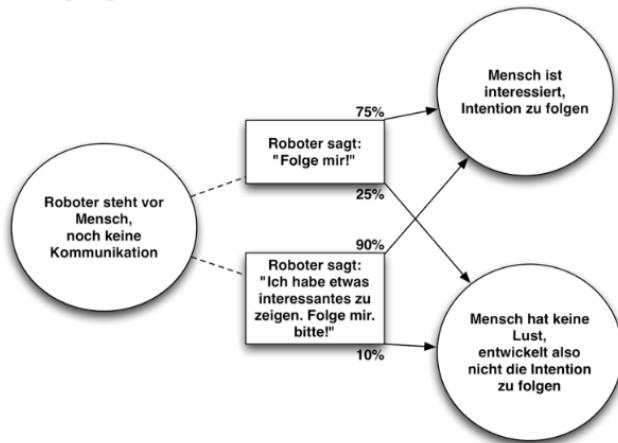


Abbildung 54: Verschiedene Auslöser haben unterschiedliche Übergangswahrscheinlichkeiten. Durch die Wahl der Handlung kann der Agent die Übergangswahrscheinlichkeiten steuern.

Struktur

- Eine Menge von symbolischen **Zuständen** der Welt S (d.h. die Umwelt bei einem MDP ist diskret)
- Eine Menge von symbolischen Handlungen des Agenten U
- Ein **Übergangsmodell** (transition model) $T(s', u, s)$ welches die stochastischen Übergänge modelliert, also das Verhalten der Welt aufspannt (modelliert den Markov-Prozess). Hierbei ist die Frage, wie wahrscheinlich ein bestimmtes Ergebnis folgt, wenn in einem beliebigen Zustand eine beliebige Handlung ausgeführt wird. Das Modell muss vollständig definiert werden (d.h. Unmöglichkeiten haben die Wahrscheinlichkeit 0).
- Ein **Belohnungsmodell** (reward model) $R(s, u)$ welches die Ziele und Absichten des Agenten modelliert. Auch negative Belohnungen/Strafen möglich.
- Ein **Startzustand** s_0

Zustände und Aktionen müssen dabei einzigartig sein. Es gilt außerdem die Markov-Eigenschaft, d.h. alle Wahrscheinlichkeiten hängen immer nur vom aktuellen Zustand ab und nicht von den bereits vergangenen. Die in jedem Schritt aufaddierten Belohnungen werden *Utility* genannt und die Aufgabe des Agenten besteht darin, Aktionen mit hoher Belohnung zu wählen, um die Utility zu maximieren.

Statische Eigenschaften

- In einem MDP ist die Welt zu jedem Zeitpunkt in **genau einem** Zustand (und nur in diesem, kein fuzzy Zustand, keine Wahrscheinlichkeits-Verteilung über mehrere Zustände)
 - brauche in echter Umgebung Diskretisierungsverfahren für kontinuierlichen Zustandsraum, im Zweifelsfall führe einen Restzustand „other“ ein
- In einem diskreten MDP ist dies ein symbolischer Zustand, welcher alle Eigenschaften der Welt repräsentiert
- Der Agent kann in jedem Agentencyklus **genau eine** Handlung ausführen
- In einem diskreten MDP (diskrete Zeit) ist dies eine symbolische Handlung (kann auf tieferer Ebene aus mehreren Subhandlungen bestehen)
- Im reinen MDP nimmt der Agent den aktuellen Zustand **immer und eindeutig** wahr

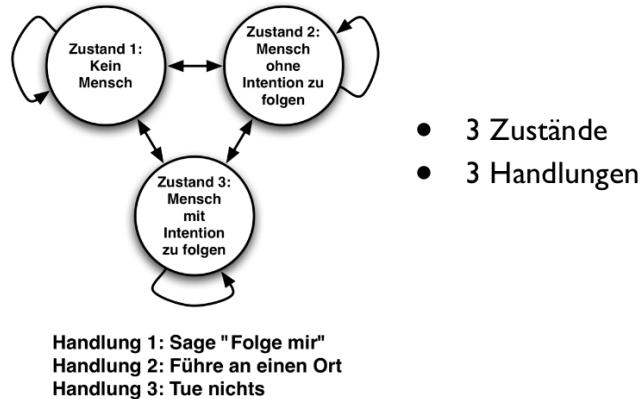


Abbildung 55: MDP Beispiel – statische Eigenschaften; hier für einen Museumsroboter welcher z.B. Möglichkeiten für Tracking und Dialogerkennung zur Aufbereitung des abstrakten Zustands hat (wenn der Mensch etwas gesagt hat hat er eher die Intention zu folgen)

Transitionsmodell

- $T(s', u, s) = p(\text{transition})$: wenn der alte Zustand s war und der Agent die Handlung u ausgeführt hat, dann beschreibt dieser Eintrag im Modell die Wahrscheinlichkeit, dass der neue Zustand s' sein wird
- auch Fehleranfälligkeit und Einschränkungen in den Entscheidungsmöglichkeiten können durch ein stochastisches Übergangsmodell modelliert werden
- Das Transitionsmodell ist ein Tensor 3. Stufe
- In der Praxis oft als Vektor (über U) von Matrizen (S, S') dargestellt: $|T| = |S| * |S| * |U|$ (siehe Abbildung 56)
- Die Größe des Modells wächst also kubisch (MDP hat klassischerweise bis zu 10.000 Zustände wobei die Mehrheit Wahrscheinlichkeit 0 hat; diese werden nicht mit modelliert, da es sonst sehr sperrig werden würde)
- Beispiel Museumsroboter: Ein Mensch erscheint im Szenario mit 25% Wahrscheinlichkeit, völlig unabhängig vom Roboterverhalten (z.B. kommt im Durchschnitt an einem durchschnittlich besuchten Tag alle 30s ein Mensch vorbei). Das Verschwinden des Menschen ist abhängig vom Roboterverhalten
 - Zustände: $S_1 = \text{Kein Mensch}$, $S_2 = \text{Mensch, keine Intention zu folgen}$, $S_3 = \text{Mensch, Intention zu folgen}$
 - Handlungen: $U_1 = \text{Sage „Folge mir!“}$, $U_2 = \text{Führe an einen Ort}$, $U_3 = \text{Tue nichts}$
 - $|T| = |S| * |S| * |U| = 3 * 3 * 3 = 27$ Einträge
- Modellbildung nicht trivial und ohne frequentistische Experimente schwierig, Handlungen können Zustände verschiedenartig beeinflussen...

U_1 (Sage "Folge mir!")			U_2 (Führe an einen Ort)			U_3 (Tue nichts)					
	S'_1	S'_2	S'_3		S'_1	S'_2	S'_3				
$S_1 :$	0.75	0.25	0.0	$S_1 :$	0.75	0.25	0.0	$S_1 :$	0.75	0.25	0.0
$S_2 :$	0.0	0.25	0.75	$S_2 :$	0.25	0.75	0.0	$S_2 :$	0.25	0.75	0.0
$S_3 :$	0.25	0.25	0.5	$S_3 :$	0.50	0.25	0.25	$S_3 :$	0.25	0.50	0.25

Abbildung 56: MDP Beispiel – Transitionsmodell, Zeilen müssen Summe 1 haben, Spalten nicht (da Bias zu manchen Zuständen)

Beloohnungsmodell

- $R(s, u) =$ Belohnung oder Strafe: wenn der Zustand s war und der Agent führt die Handlung u aus, erhält er die sofortige Belohnung (oder negative Belohnung, also Strafe) wie vom Modell an $R(s, u)$ gegeben
- Das Belohnungsmodell ist eine Matrix
- Die Einträge können beliebige Realzahlen sein, je nach Modellierung
- Durch das Belohnungsmodell können auch konkurrierende oder ergänzende Zielsetzungen modelliert werden
- Beispiel Museumsroboter: *Sprechen* und *Führen* kostet immer -1 Strafe, verglichen mit *Tue nichts*, denn der Roboter ist faul und möchte Energie sparen, außerdem soll unnötige Unruhe im Museum vermieden werden. Einen folgenden Menschen an einen Ort zu führen gibt +5 Belohnung.

	U_1	U_2	U_3
$S_1 :$	-1	-1	0
$S_2 :$	-1	-1	0
$S_3 :$	-1	4	0

Abbildung 57: MDP Beispiel – Belohnungsmodell

Ziel des Agenten

- Das Ziel des rationalen Agenten im MDP ist, die Summe der Belohnungen bis zu einem bestimmten Zeithorizont (im reinen MDP meistens unendlich) zu maximieren
- Dies führt zu einer **Gewinnmaximierung** bzw. **Risikoabschätzung** in die Zukunft als Grundlage für den Entscheidungsprozess (zwischendurch negative Belohnung einzusammeln kann Sinn machen, Kosten in Kauf nehmen um langfristig große Belohnung einzusammeln; im Entscheidungsprozess müssen viele potentielle Handlungsketten gleichzeitig berücksichtigt werden)
- Das einzige explizite Ziel des Agenten ist die Gewinnmaximierung
- Sie bezieht implizit alle gegebenen Zielsetzungen in den Entscheidungsprozess mit ein

Gewinnmaximierung

- Da in einer stochastischen Welt keine festen Handlungsketten (in die Zukunft) existieren, wird eine Entscheidungsfunktion (*policy*) berechnet
- Die Entscheidungsfunktion π gibt für jeden Zustand s eine Handlung zurück: $\pi(s)$ (im klassischen MDP ist Entscheidungsfunktion eine Look-Up-Table)
 - deswegen wird in diesem Kontext meistens von „entscheiden“ nicht wie z.B. in STRIPS von „planen“ geredet (Planung ändert sich zu jedem Schritt)
- Ein Agent in einem MDP muss nur eine optimale Entscheidungsfunktion π^* berechnen, welche immer die Handlung mit der größten, langfristigen Gewinnerwartung wählt: $\pi^*(s)$
- Es gibt immer *mindestens* eine $\pi^*(s)$, welche zu einem gegebenen MDP Modell optimale Entscheidungen liefert (selbst wenn es nur schlechte Möglichkeiten gibt und in der Praxis keinen Sinn macht!)
- Bei unendlichem Horizont ist die optimale Entscheidungsfunktion stationär: es gibt für einen Zustand immer dieselbe optimale Handlung. In einem voll beobachtbaren (reinen) MDP ist die Variante mit unendlichem Horizont somit die einfachste. Im Folgenden wird bei reinen MDPs nur der Fall des unendlichen Zeithorizonts berücksichtigt

Utility

- Die Utility einer Zustandskette ist die Summe ihrer Belohnungen bzw. Strafen:

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$
- Die Einführung eines Abschlagsfaktors (*discount*) $\gamma \in [0, 1]$ auf Zustände in der Zukunft ermöglicht eine endliche utility für alle unendlichen Zustandsketten (wie beim Menschen, Belohnung morgen ist mehr wert als Belohnung in einem Jahr):

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

 Somit bildet sich eine Reihe welche asymptotisch konvergiert und das Problem der unendlichen Summe ist gelöst. γ sollte aber nicht zu aggressiv gewählt werden

Entscheidungsfunktion

- Es existieren verschiedene Verfahren, um eine optimale Entscheidungsfunktion aus dem gegebenen MDP Modell (S, A, T, B) zu berechnen. Unter anderem: Policy Iteration oder Reinforcement Learning einer Policy (Gefahr hierbei: „blinde Flecken“ der Policy)
- Das wichtigste aber ist: **Value Iteration** (zur Berechnung der optimalen Policy, bietet sich nur nicht an wenn Modellbildung im Vorfeld für Szenario zu schwierig ist)

5.2.2 MDP Value Iteration

Problem

- Für jeden Zustand wird eine utility berechnet, die die erwartete utility aller möglichen, folgenden Zustandssequenzen widerspiegelt
- Daraus kann die Entscheidungsfunktion direkt gewonnen werden
- Die Zustandssequenzen wiederum hängen von der Entscheidungsfunktion ab
- \Rightarrow verschränktes Problem

Ansatz

- Iterativer Ansatz
- Die wirkliche utility eines Zustands wäre die utility bezogen auf die optimale policy

$$U_{real}(s) = U^{\pi^*}(s)$$
- Die wirkliche utility ist jedoch zu Beginn nicht bekannt, da die optimale policy noch berechnet werden muss
- Zuerst wird eine beliebige policy π_{start} gewählt
- Dazu wird jedem Zustand eine initiale utility zugewiesen
- Die tatsächliche utility jedes Zustands, bezogen auf eine policy π , ist die Summe aller zukünftigen Belohnungen:
$$U^\pi(s) = E \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, u_t) | \pi, s_0 = s \right]$$
- die Summe der zukünftigen Belohnungen hängt jedoch von der Handlungswahl und damit von π ab

Handlungswahl

- Es kann nun eine policy gefunden werden, die auf den aktuell zugewiesenen utilities basiert und I-Schritt weit optimal ist
- Nach der utility Funktion wird nun die Handlung, die die erwartete zukünftige Belohnung maximiert gewählt: $\pi^{1step*}(s) = \operatorname{argmax}_u \sum_{s'} T(s', u, s)U(s')$
- Durch diese Formel wird auch ein Zusammenhang zwischen einem Zustand und seinem Nachbarn gebildet

Iterationsschritt

- Nun wird die Zuweisung wieder in die utility-Berechnung zurückgeführt
- Es entsteht eine Iterationsformel zwischen benachbarten Zuständen: $U(s) = \gamma \max_u (R(s, u) + \sum_{s'} T(s', u, s)U(s'))$ = Bellmann Formel
- Somit kann rückwärts von $U(s)$ auf $U(s')$ geschlossen werden

Iterationsablauf

- Beginn: irgendwelche niedrigen Werte für die utilites aller Zustände
- Iterationsschritt: Anwendung der Bellmann Formel auf jeden Zustand unter Nutzung aller seiner Nachbarn
- Vielfache Anwendung des Iterationsschritts auf alle Zustände
- Irgendwann wird ein Gleichgewicht erreicht, das die optimale policy π^* ist
- Die Iteration konvergiert zur optimalen policy
- In der Praxis werden die Iterationen bis zu einer akzeptablen Konvergenzschanke durchgeführt

Algorithm 4 MDP discreteValueIteration()

```

1: repeat
2:   for each state  $s$  in  $S$  do
3:      $U'(s) \leftarrow \gamma \max_u (R(s, u) + \sum_{s'} T(s', u, s)U(s'))$ 
4:   end for
5: until convergence bound                                 $\triangleright$  Änderung der utility-Werte zwischen zwei Schritten
6: return  $U$ 

```

Algorithmus Beispiel Museumsroboter:

- Zustände: $S_1 = \text{Kein Mensch}$, $S_2 = \text{Mensch, keine Intention zu folgen}$, $S_3 = \text{Mensch, Intention zu folgen}$
- Handlungen: $U_1 = \text{Sage } \text{„Folge mir!“}$, $U_2 = \text{Führe an einen Ort}$, $U_3 = \text{Tue nichts}$
- Initialisiere: $U(s_1 = \text{kein Mensch}) = -1$, $U(s_2 = \text{Mensch, folgt nicht}) = -1$, $U(s_3 = \text{Mensch, folgt}) = -1$; wähle $\gamma = 0.9$
- $U(s) = \gamma \max_u (R(s, u) + \sum_{s'} T(s', u, s)U(s'))$

5 Probabilistisches Entscheiden

	U_1 (Sage "Folge mir!")	U_2 (Führe an einen Ort)	U_3 (Tue nichts)
	S'_1	S'_2	S'_3
S_1 :	0.75	0.25	0.0
S_2 :	0.0	0.25	0.75
S_3 :	0.25	0.25	0.5
S_1 :	0.75	0.25	0.0
S_2 :	0.25	0.75	0.0
S_3 :	0.50	0.25	0.25
S_1 :	0.75	0.25	0.0
S_2 :	0.25	0.75	0.0
S_3 :	0.25	0.50	0.25

Belohnungsmodell:

	U_1	U_2	U_3
S_1 :	-1	-1	0
S_2 :	-1	-1	0
S_3 :	-1	4	0

Abbildung 58: MDP VI Beispiel – Modell

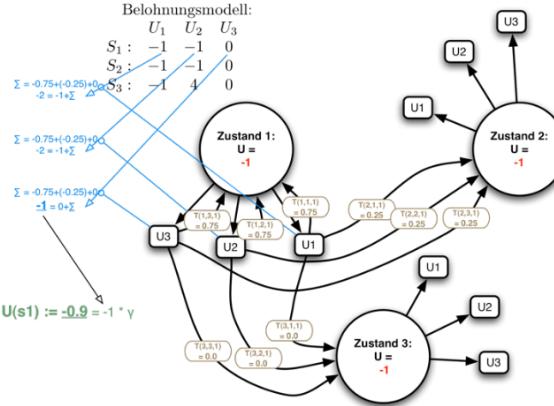


Abbildung 59: MDP VI Beispiel – Iteration

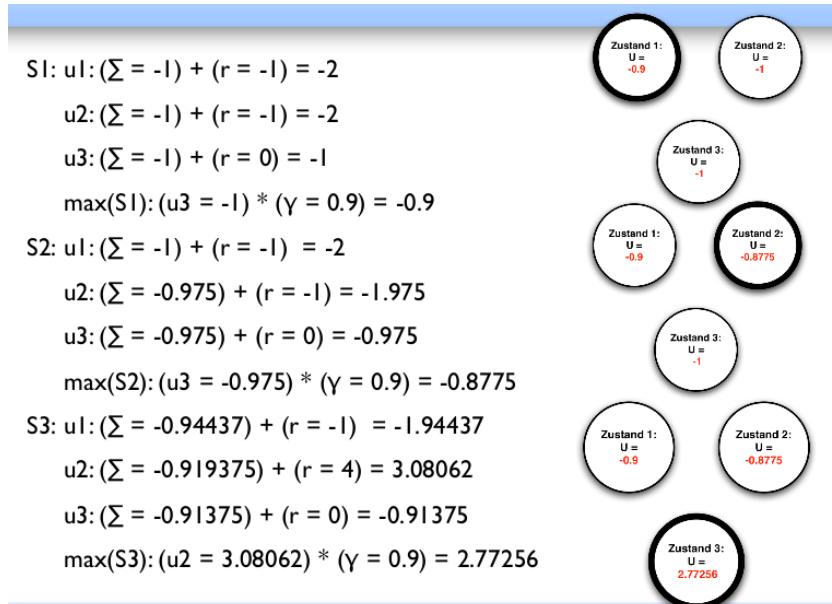


Abbildung 60: MDP VI Beispiel – Iterationsschritt

Eigenschaften

- MDP Value Iteration ist eine Kontraktion: der Algorithmus konvergiert immer (auch wenn Modell keinen Sinn macht, dies muss vorher sicher gestellt werden)
- In der Praxis konvergiert er bei kleinerem γ schneller
- MDP Value Iteration ist verhältnismäßig schnell (im Vergleich zu anderen Verfahren und anderen Arten von MDPs)
- Fast optimale Entscheidungsfunktionen sind in der Praxis ausreichend

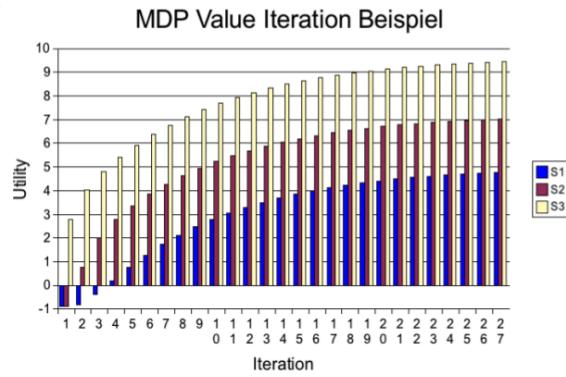


Abbildung 61: MDP VI Beispiel – Ergebnis,
 $S_1 : u_3$ am besten ab Iterationsschritt 1,
 $S_2 : u_1$ am besten ab Iterationsschritt 2
 $S_3 : u_2$ am besten ab Iterationsschritt 1

5.2.3 MDP – Schlussfolgerung

- Es gibt ein algorithmisches Konzept für optimale Entscheidungsfindung in stochastischen Umgebungen
- Ein rationaler Agent kann in Bezug auf Belohnungsmaximierung entscheiden
- Die (fast) optimale Belohnungsmaximierung ist berechenbar
- Die Modellierung eines Szenarios ist allerdings schwierig
- Nur: Welche Umgebungseigenschaften decken reine MDPs ab?

Umgebungseigenschaften Markov Decision Processes sind ein theoretisches Rahmenwerk zur Modellierung von Entscheidungsfindung in

- Dynamischen
- Sequentiellen
- Diskreten (aber auch kontinuierlichen)
- Stochastischen
- **Vollständig** beobachtbaren (in Praxis meistens nicht gegeben, z.B. bei Serviceroboter)
- (Multiagent-)

Domänen.

Es gibt Erweiterungen von MDPs für kontinuierliche Umgebungen (Zustände, Handlungen). Es gibt eine Erweiterung von MDPs für unvollständig beobachtbare Umgebungen: **Partially Observable Markov Decision Processes**.

5.3 POMDP (Partially Observable Markov Decision Processes)

Eine Erweiterung von MDPs für unvollständig beobachtbare Umgebungen. Perzeption ist üblicherweise verrauscht und deckt nur einen Teil des Szenarios ab. Beispielsweise Spracherkennung ist eine zentrale Fähigkeit für Serviceroboter (natürliche Mensch-Roboter-Interaktion), Sprache ist aber sehr verrauscht.

Zustandsschätzung

- Die Schätzung des aktuellen Zustands muss nicht nur auf der Perzeption basieren
- Grundprinzip: rationaler Agent in Domäne mit Markov Eigenschaft:
 - Aktueller Zustand kann über unvollkommene Perzeption beobachtet werden (measurement)
 - Aktueller Zustand kann auf Basis des vorherigen Zustands und der ausgeführten Handlung vorhergesagt werden (prediction)
- Die **Beobachtung** wird durch ein auf die Eigenschaften des physischen oder logischen Sensors abgestimmtes Unsicherheitsmodell abgebildet
- Die Unsicherheit kann so explizit, numerisch angegeben werden
- Diese Unsicherheit kann verschieden modelliert werden:
 - kontinuierlich oder diskret
 - parametrisch oder nicht-parametrisch
- Die **Vorhersage** wird über ein Transitionsmodell realisiert
- Das Transitionsmodell führt zu einer Wahrscheinlichkeitsverteilung über allen Zuständen wie im MDP
- Eine Vorhersage funktioniert nur iterativ, ausgehend von einem aktuellen Zustand (bzw. begrenzten Historie: Markov Eigenschaft)

Bayes Filter

Vorhersage und Beobachtung können nun fusioniert werden, um zusammen eine höhere Genauigkeit zu erreichen!

Idee:

- Eine gute Vorhersage kann eine sehr ungenaue Beobachtung verbessern
- Eine gute Beobachtung präzisiert jedoch eine ungenaue Vorhersage
- Da die Vorhersage nur iterativ funktioniert, arbeitet der Bayes Filter ebenfalls iterativ von Zeit $t - 1$ nach t

Zustandsvermutung:

- Die Ausgabe einer Vorhersage und auch einer Beobachtung ist eine Wahrscheinlichkeitsverteilung
- Da der Bayes Filter iterativ läuft, ist die Eingabe also auch eine Wahrscheinlichkeitsverteilung
- Der Agent besitzt daher zu jedem Zeitpunkt eine Zustandsvermutung (belief)
- Die Zustandsvermutung repräsentiert die subjektive Wahrnehmung des Agenten bezüglich der nur unvollständig beobachtbaren Welt (Zustandsraum)
- Die Zustandsvermutung ist eine
 - diskrete Wahrscheinlichkeitsverteilung über einem diskreten Zustandsraum
 - oder eine kontinuierliche Verteilung über einem kontinuierlichen Zustandsraum

Vorhersage:

- Zustände x , Aktionen u
- Diskreter Fall: $\bar{p}_{k,t} = \sum_i p(X_t = x_k | u_t, X_{t-1} = x_i) p_{i,t-1}$

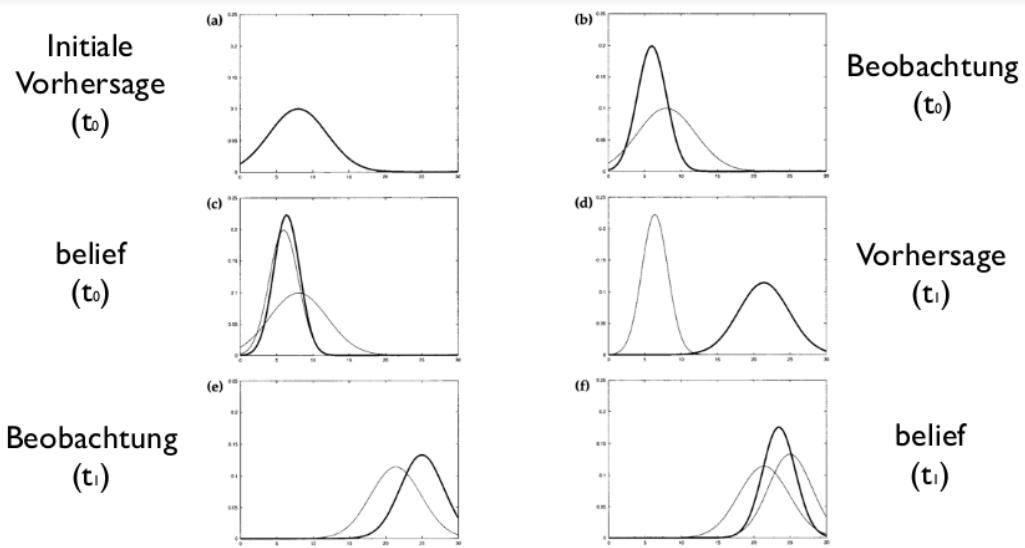


Abbildung 62: Beispiel – kontinuierlicher Bayesfilter

- Kontinuierlicher Fall: $\bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1}$

Beobachtung:

- Zustände x , Beobachtung z
- Diskreter Fall: $p_{k,t} = \eta p(z_t|X_t = x_k)\bar{p}(k,t)$
- Kontinuierlicher Fall: $bel(x_t) = \eta p(z_t|x_t)\bar{bel}(x_t)$

Diskreter Bayes Filter:

Abbildung 63

```
discreteBayesFilter( $P_{t-1}, u_t, z_t$ ):
    for each  $p_k$  in  $P$  do
         $\bar{p}_{k,t} = \sum_i p(X_t = x_k|u_t, X_{t-1} = x_i)p_{i,t-1}$ 
         $p_{k,t} = \eta p(z_t|X_t = x_k)\bar{p}(k,t)$ 
    endfor
    return  $P_t$ 
```

(a) Vollständiger diskreter Bayes Filter mit Zuständen x , Aktion u und Beobachtung z

```
discreteBayesFilter( $S_{t-1}, u_t, z_t, T, O$ ):
    for each  $s_t$  in  $S$  do
         $p(s_t) = \alpha O(z_t, s_t) \sum_{s_{t-1}} T(s_t, u_t, s_{t-1}) p(s_{t-1})$ 
    endfor
    return  $s_t$ 
```

(b) Alternative Darstellung mit Normalisierer α , Zuständen s , explizitem Transitionsmodell T und Beobachtungsmodell O

Abbildung 63: Diskreter Bayes Filter

Kontinuierlicher Bayes Filter:

Abbildung 64 Kontinuierliche Bayes Filter werden durch spezielle Methoden realisiert.

```
generalBayesFilter( $bel_{t-1}, u_t, z_t$ ):
    for each  $x_t$  in  $bel$  do
         $\bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1}$ 
         $bel(x_t) = \eta p(z_t|x_t)\bar{bel}(x_t)$ 
    endfor
    return  $bel(x_t)$ 
```

Abbildung 64: Allgemeine kontinuierlicher Bayes Filter

Parametrische Filter:

- Kalman Filter
- Extended Kalman Filter
- Unscented Kalman Filter
- Information Filter
- Extended Information Filter

Nicht-parametrische Filter: Partikelfilter

MDP + Bayes Filter:

Um MDPs in einer unvollständig beobachteten Umgebung nutzen zu können werden Bayes Filter mit MDPs verbunden. Das erfordert die Einführung der Zustandsvermutung in den MDP. Dadurch hat der MDP zwei Komponenten:

- Realer Zustand der Welt
- Subjektive Zustandsvermutung des Agenten

⇒ POMDPs

5.3.1 Eigenschaften und Ablauf von POMDPs

POMDPs ermöglichen die Entscheidungsfindung in Domänen mit den Eigenschaften:

- Dynamisch
- Sequentiell
- Diskret oder Kontinuierlich
- Stochastisch
- Unvollständig beobachtbar

Struktur/Modell

- Eine Menge von symbolischen **Zuständen** der Welt S
- Eine Menge von symbolischen **Handlungen** des Agenten U
- Eine Menge von symbolischen **Beobachtungen** des Agenten M
- Ein **Übergangsmodell** (transition model) $T(s', u, s)$ welches die stochastischen Übergänge modelliert
- Ein **Belohnungsmodell** (reward model) $R(s, u)$ welches die Ziele und Absichten des Agenten modelliert
- Ein **Beobachtungsmodell** (observation model) $O(m, s')$ welches die Ungenauigkeit der Wahrnehmung des Agenten modelliert
- Einen Startzustand s_0

Statische Eigenschaften

- Die statischen Eigenschaften des reinen MDP gelten auch im POMDP
- Zusätzlich gilt: Der Agent kann in jedem Agentenzyklus genau eine Beobachtung machen
- Weiterhin: In einem diskreten POMDP ist dies eine symbolische Beobachtung
- Im Folgenden wird von diskreten POMDPs ausgegangen

Dynamikmodelle

- Transitions- und Belohnungsmodell funktionieren wie beim MDP
- Hinzu kommt das Beobachtungsmodell:
 - $O(m, s')$ = Wahrscheinlichkeit: wenn die Beobachtung m gemacht wurde, dann ist die Wahrscheinlichkeit, dass der wirkliche Weltzustand s' ist, vom Modell an $O(m, s')$ gegeben
 - Das Beobachtungsmodell ist eine Matrix

Sekundäre Eigenschaften

- Die Entscheidungsfunktion (policy) existiert auch beim POMDP, wenn auch in anderer Form
- Hinzu kommt die Zustandsvermutung:
 - Im diskreten Fall eine diskrete Wahrscheinlichkeitsverteilung über alle Zustände
 - Die Zustandsvermutung wird zu jedem Zeitschritt durch Bayes Filterung neu berechnet
 - Führt zu einer klaren Trennung zwischen objektiver Realität (echter Weltzustand) und der subjektiven Sicht des Agenten

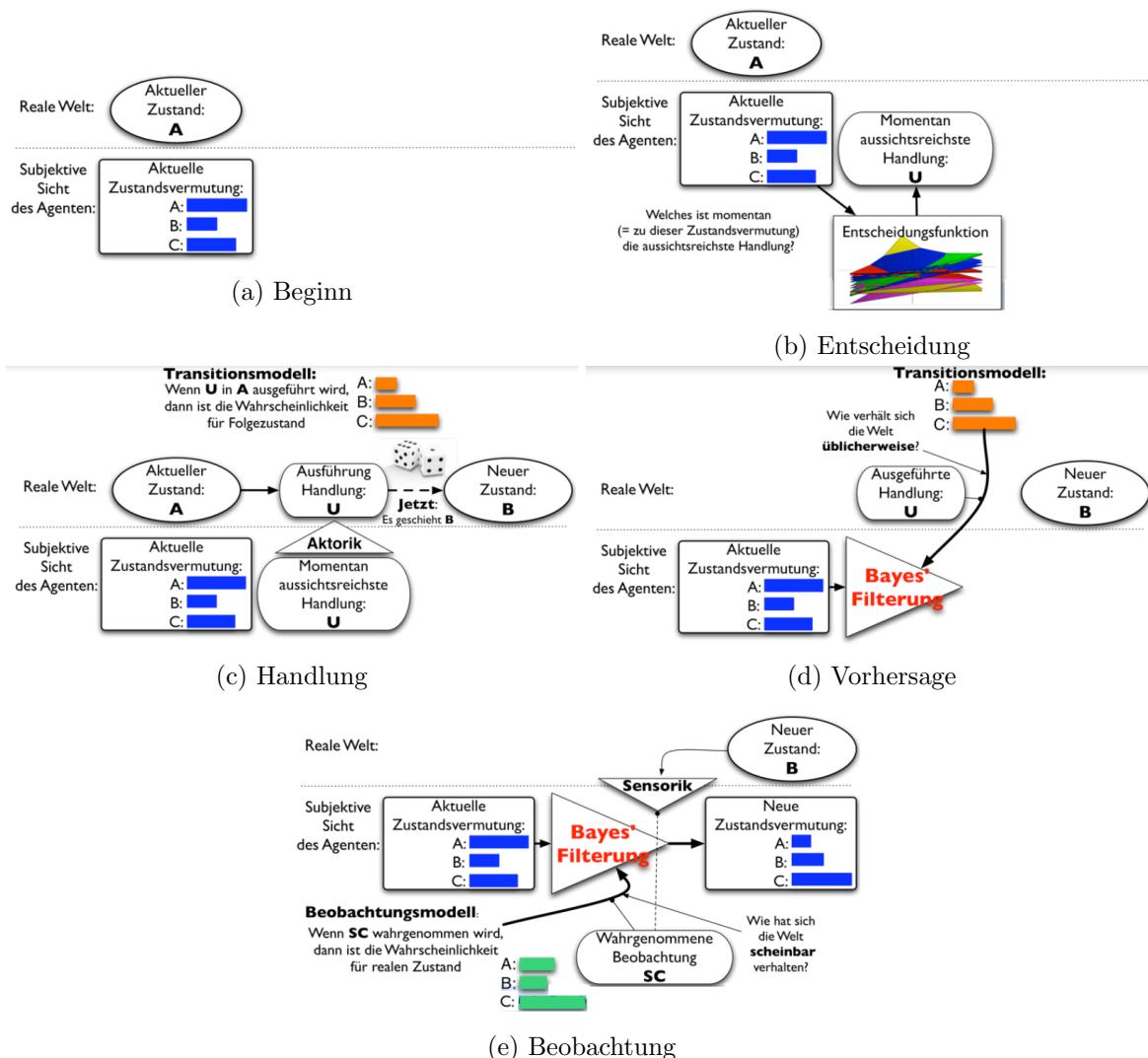


Abbildung 65: POMDP – Schritte

Entscheidungsfunktion

- Bei MDPs enthält die Entscheidungsfunktion eine Handlung zu jedem Zustand
- Im Gegensatz zu MDPs muss die Entscheidungsfunktion bei POMDPs über **alle möglichen Zustandsvermutungen** definiert sein
- Struktur der Entscheidungsfunktion
 - Bei diskreten POMDPs: über dem $|S| - 1$ dimensionalen Raum aller möglichen Zustandsvermutungen definiert
 - Bei kontinuierlichen POMDPs: über einem unendlich-dimensionalen Raum definiert, daher speziell über endlich-dimensionalen Teilräumen angegeben
- Berechnung der Entscheidungsfunktion: aufgrund der komplexen Struktur ist die Berechnung deutlich komplizierter als bei MDPs (Value Iteration existiert, ist aber anders)

Raum der Vermutungen (diskrete POMDP)

- Alle möglichen Wahrscheinlichkeiten über allen Zuständen s eines Szenarios spannen den Raum der Zustandsvermutungen auf
- Da $\sum p(s_i) = 1$, ist der Raum begrenzt, also ein Simplex (= konvexe Hülle von $n + 1$ Punkten in einem n -dimensionalen Raum)
- Wegen der Summe $|S| - 1$ Dimensionen
- Beispiele in Abbildung 66

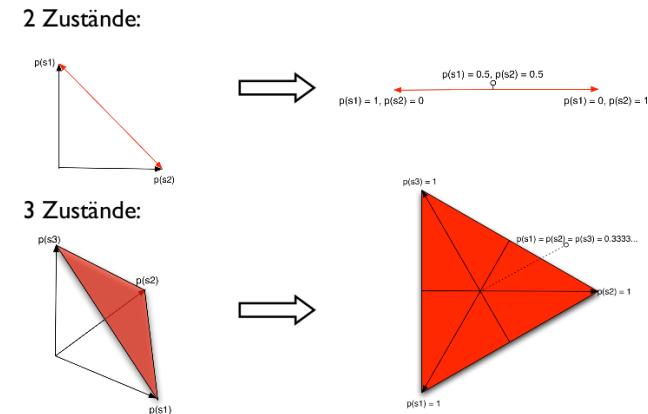


Abbildung 66: Beispiel – Raum der Vermutungen

5.3.2 POMDP Value Iteration

Aufgrund der komplexeren Struktur ist die Berechnung der Entscheidungsfunktion deutlich komplizierter als bei MDPs. Value Iteration existiert auch für POMDPs, funktioniert jedoch anders:

- Iterativer Ansatz (wie bei MDP)
- Ein endlicher Horizont wird vorausgesetzt
- Ein schrittweises Vorausschauen wird von einer initialen Entscheidungsfunktion aus durchgeführt
- Das Vorausschauen muss die stochastische Natur der Weltdynamik, als auch die verschiedenen genauen Beobachtungen eben dieser Dynamik in die Abwägung einfließen lassen

Struktur

- Lineare Funktionen α repräsentieren die utility kontinuierlich über dem gesamten Zustandsvermutungs-Simplex bezogen auf eine Handlung (nicht Zustand, wie im MDP)
- Für jede Zustandsvermutung kann aus den Funktionen eine utility berechnet werden
- Beispiele (Abbildung 67):

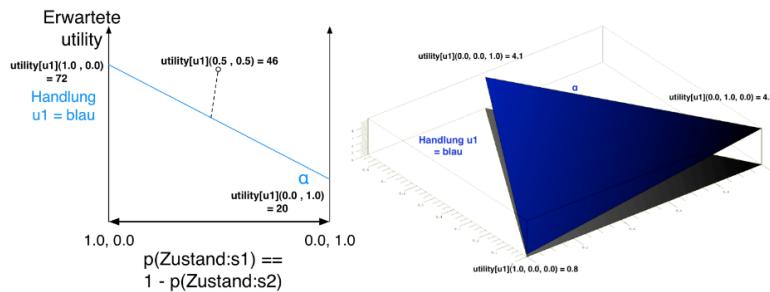


Abbildung 67: POMDP VI Struktur – Beispiele für 2 und 3 Zustände

Beginn

- Der Algorithmus beginnt mit dem Pseudohorizont 0
- Nun wird bis zu einem Horizont T in die Zukunft iteriert
- Dabei wird eine Menge Γ von linearen Funktionen α erzeugt, die Gewinnerwartungswerte für Mengen möglicher Handlungsketten darstellen

Schritt besteht aus zwei wesentlichen Teilen

1. Temporäre Koeffizienten werden durch die Bayes Vorwärtsfilterung (Projektion) jeder vorigen linearen Funktion über jede mögliche Handlung und Wahrnehmung erzeugt
 - Die Projektion jeder linearen Funktion α aus Γ in $t - 1$ durch jede Kombination von Aktion u und Beobachtung m führt zu den temporären Koeffizienten $v\{k\}$ (siehe Code in Abbildung 68)
2. Die temporären Koeffizienten werden für die Bildung der Erwartung über alle möglichen Beobachtungskombinationen verwendet, woraus die neuen linearen Funktionen erzeugt werden
 - Aus den temporären Koeffizienten wird für jede mögliche Kombination von Beobachtungen die Gewinnerwartung berechnet und eine neue lineare Funktion α' erzeugt (siehe Code in Abbildung 69)

```

for each  $\alpha^k$  in  $\Gamma$  do
    for each  $u$  in Actions do
        for each  $m$  in Measurements do
            for  $j = 1$  to  $|\text{States}|$  do
                 $v_{a,m,j}^k = \sum_i^{|\text{States}|} \alpha_i^k * O[m, i] * T[i, u, j]$ 
            endfor
        endfor
    endfor
endfor

```

Abbildung 68: POMDP VI Schritt 1

```

for each  $a$  in Actions do
    for each  $mset[*] = (1_1, 1_2, \dots, 1_M)$  to  $(|\Gamma|_1, \dots, |\Gamma|_M)$  do
        for  $j = 1$  to  $|\text{States}|$  do
            for each  $m$  in Measurements do
                 $v_j'' += v_{a,m,j}^{mset[m]}$ 
            endfor
             $\alpha'_a[j] = \gamma * (R(j, a) + v_j'')$ 
        endfor
         $\Gamma' \leq \alpha'_a$ 
    endfor
endfor

```

Abbildung 69: POMDP VI Schritt 2

Entscheidungsfunktion

- Die Entscheidungsfunktion ist das Maximum der utility in der Menge von linearen Funktionen über dem jeweiligen Bereich des Zustandsvermutungsraums
- Diese Menge ist immer konvex über dem Zustandsvermutungs-Simplex

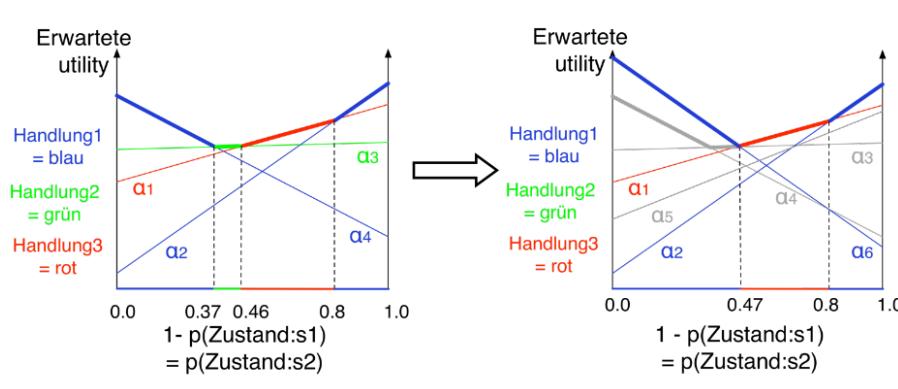


Abbildung 70: POMDP VI Schema; 2 Zustände, 3 Handlungen

Komplexität

- Schritt Teil 2 ist aus Komplexitätstheoretischer Sicht bedeutend:
`for each $mset[*] = (1_1, 1_2, \dots, 1_M)$ to $(|\Gamma|_1, \dots, |\Gamma|_M)$ do`
- Hier geschieht eine kombinatorische Explosion bezüglich der Anzahl der erzeugten α :
 $|\Gamma_t| = |\text{Actions}| * |\Gamma_{t-1}|^{|\text{Observations}|}$
- Was zu einer doppelt-exponentiellen Komplexität (2-EXPSPACE) führt:
 $|\Gamma| = |\text{Actions}|^{(\sum_{i=0}^{t-1} |\text{Observations}|^i)}$

- Beispiel 3 Aktionen, 3 Beobachtungen, $|\Gamma|$:

- Horizont 1 : 3
- Horizont 2 : 81
- Horizont 3 : 1594323
- Horizont 4 : 1.2×10^{19}
- Horizont 5 : 5.4×10^{57}

⇒ exakte POMDP Value Iteration unbrauchbar!!

- Um die Komplexität handhabbar zu machen, wurden **approximative Value Iteration** Verfahren entwickelt. Dadurch wird meistens der Teil 2 des Iterationsschritts entschärft. Alle relevanten Verfahren haben garantierte Fehlerschranken und zeichnen sich durch sehr gute Approximation aus

Point Based Value Iteration als approximative Value Iteration

- Nutzt ein dynamisches Raster zur Berechnung der Entscheidungsfunktion
- Die Entscheidungsfunktion wird nur für eine dynamische Menge von Stellvertreterpunkten im Zustandsvermutungsraum berechnet
- Sie gilt jedoch für den gesamten Raum
- Schema: Die Genauigkeit hängt in der Praxis von der Anzahl der Vertreterpunkte und der Änderungshäufigkeit optimaler Handlungen über dem Zustandsraum ab (vgl. Abbildung 71)

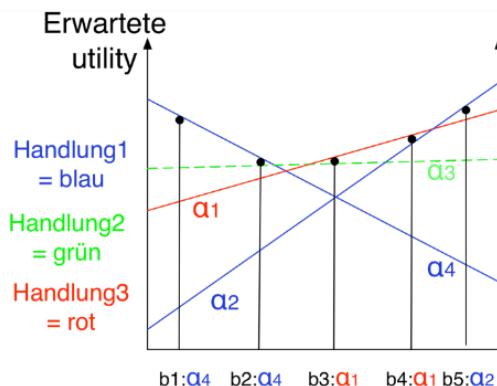


Abbildung 71: PB VI Schema

Weitere approximative Value Iterations sind PERSEUS, HSV12, SARSOP..

5.3.3 POMDPs und Serviceroboter

- Die Nutzung von POMDPs in autonomen Servicerobotern steht noch ganz am Anfang
- Momentan fast nur in Simulationen und virtuellen Benchmarkszenarien im Einsatz
- Mono-Modale SLAM¹³-Szenarien stehen dort noch im Mittelpunkt
- Die Übertragung der Theorie auf Robotersteuerungen wird gerade erforscht. Abbildung 72 zeigt eine Einordnung aktueller Technologien.
- Herausforderungen

¹³Simultaneous Localization And Mapping (SLAM) ist eine zentrale Fähigkeit für autonome, mobile Roboter, also auch Serviceroboter

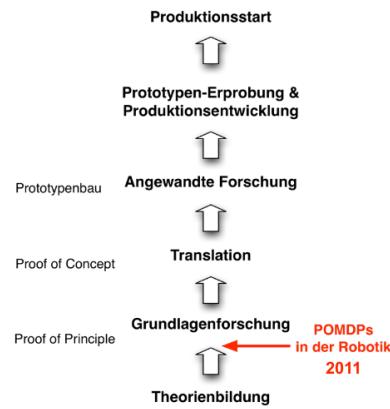


Abbildung 72: POMDPs – Technologieeinordnung

- Überwindung der Abstraktionslücke zwischen Roboter Sensorik/Aktorik und symbolischen POMDPs
- stochastische Modellierung von Szenarien
- Strukturierung der stochastischen Modelle
- Integration der Stärken klassischer Planung/Ausführung
- Integration von maschinellem Lernen
- POMDPs in einer Robotersteuerung (vgl. Abbildung 73)

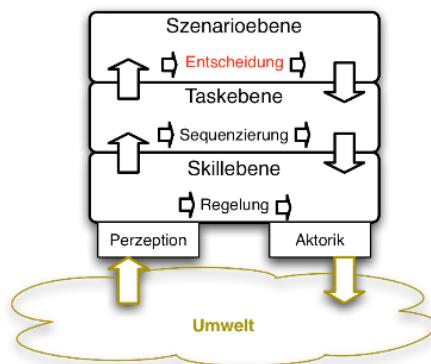


Abbildung 73: POMDPs in einer Robotersteuerung

- POMDPs werden mit anderen Steuerungsmethoden zusammen eingesetzt
- Schichtung trennt die Methoden