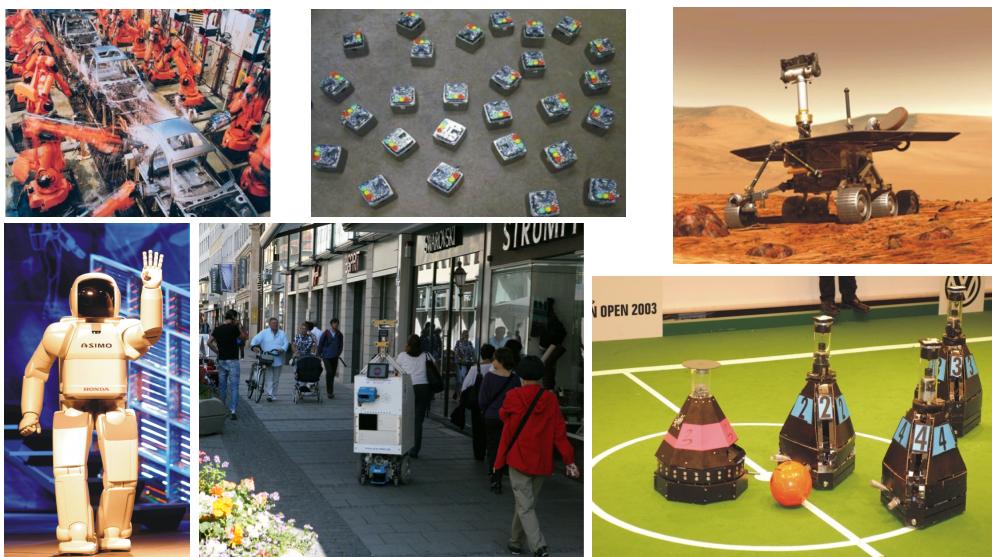


Technik Autonomer Systeme: Multi Agent Systems Architekturen

Dirk Wollherr

*Lehrstuhl für Steuerungs- und Regelungstechnik
Technische Universität München*

Autonome Systeme



Bis Jetzt...

- Control
 - Sensoren & Aktuatoren
 - Lokalisierung & Kartenerstellung
 - Pfadplanung
 - Vision & Blickrichtungssteuerung
- ⇒ Wichtige Bausteine autonomer Systeme

3

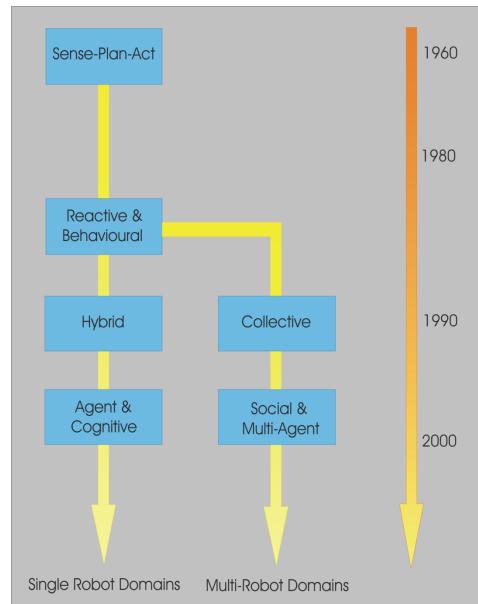
Problembeschreibung

- Kombination der Komponenten auf systematische Weise
 - Hinsichtlich welcher Ordnung?
 - Mit welcher Priorität?
- Wie kreiert man komplexe, intelligente, robuste, adaptive Roboter?
- Wie befasst man sich mit:
 - unbekannten Umgebungen
 - dynamischen Umgebungen
 - fehlerhaften Sensoren & Aktuatoren
 - Zeitbeschränkungen

4

Roboter-Architekturen

- Roboter-Architekturen spezifizieren eine Anordnung von Komponenten und deren Interaktion
- Roboter-Architekturen beinhalten eine Reihe von Regeln zur Organisation des Systems:
 - bzgl. der Struktur
 - Beschränkungen



5

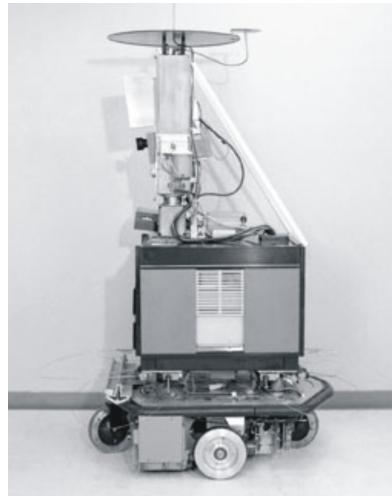
Überblick

- Roboter-Architekturen
 - **Sense-Plan-Act (Deliberative)**
 - Reaktive Architekturen
 - Verhaltensorientierte Architekturen
 - Hybride Architekturen
- Agenten
 - Rationale Agenten
 - Theorie des Nutzens – Utility Theory
 - Markov Entscheidungsprozesse

6

Sense-Plan-Act (Deliberative)

- Erste Implementierungen in 60er Jahren
(Shakey, Artificial Intelligence Lab, SRI International)
- Einfache sequenzielle Schritte
 - Sensordaten sammeln (S)
 - Daten verarbeiten → Plan erstellen (P)
 - Aktionen durchführen (A)
- Nachteile
 - Zeit
 - ◆ benötigt Suche (Search) → langsam
 - ◆ in dynamische Umgebungen: veraltete Pläne
 - Voraussetzung, dass Umgebungsrepräsentation
 - ◆ komplett
 - ◆ präzise
 - ◆ aktuell
 - ◆ vorhersehbar
 - Voraussetzung, dass Durchführung
 - ◆ sequenziell
 - ◆ zuverlässig



VIDEO

7

Überblick

- Roboter-Architekturen
 - Sense-Plan-Act (Deliberative)
 - **Reaktive Architekturen**
 - Verhaltensorientierte Architekturen
 - Hybride Architekturen
- Agenten
 - Rationale Agenten
 - Theorie des Nutzens – Utility Theory
 - Markov Entscheidungsprozesse

8

Reaktive Architekturen

- Erster Ansatz: Subsumption Architecture (Brooks '86, MIT)

- Eigenschaften:

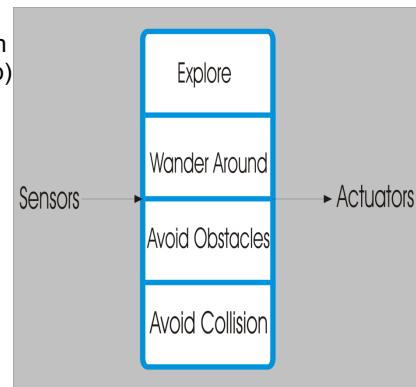
- keine Umgebungsrepräsentation benötigt
- alle Regeln können gleichzeitig durchgeführt werden
- Komponenten sind in Ebenen organisiert (bottom up)
- untere Ebenen erledigen lower-level Aufgaben
- höhere Ebenen können die Ausgaben von niedrigeren Ebenen überschreiben um deren Funktionalität zu erweitern
- dadurch können höhere Ebenen aufgesetzt werden, ohne die darunter liegenden ändern zu müssen

- Vorteile:

- Reaktivität (Geschwindigkeit, Echtzeitfähigkeit)
- inkrementelles Design
- Allgemeinheit

Nachteile:

- Kein komplexes, intelligentes Verhalten
- Keine Planung für zukünftige Aktionen



9

Genghis MIT AI Lab (Brooks), Ende der '80er



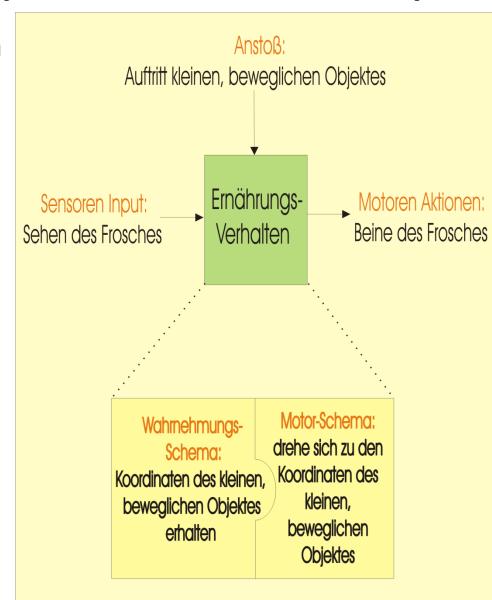
Überblick

- Roboter Architekturen
 - Sense-Plan-Act (Deliberative)
 - Reaktive Architekturen
 - **Verhaltensorientierte Architekturen**
 - Hybride Architekturen
- Agenten
 - Rationale Agenten
 - Theorie des Nutzens – Utility Theory
 - Markov Entscheidungsprozesse

11

Behavior-based Robotics (Verhaltensorientiert)

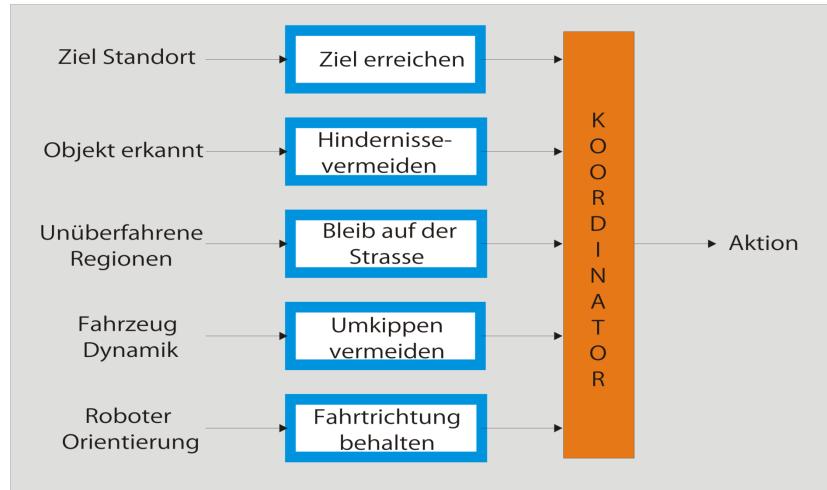
- Verhalten: Mapping von Sensoren-Eingaben zu Motoren-Aktionen, die zur Ausführung einer Aufgabe führen
- Schema:
 - Besteht aus:
 - ◆ Informationen über Wahrnehmung und Aktionsmöglichkeiten (Wissen, Datenstrukturen, Modelle)
 - ◆ Algorithmen für die Ausführung der Aufgabe
 - Allgemeines Template für Aktionsbeschreibungen
- Ausdruck von Verhalten erfolgt durch:
 - Stimulus-Response Diagramme
 - Funktionsnotation
 - Finite State Automata Diagramme



12

Aufbau von Verhalten

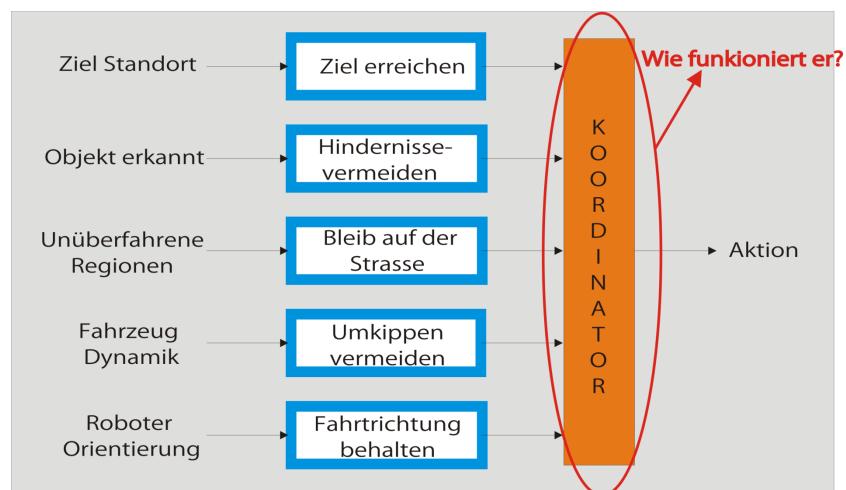
- **Problem:** Wie kombiniert man mehrere Verhaltensweisen?
- Navigations-Beispiel:



13

Aufbau von Verhalten

- **Problem:** Wie kombiniert man mehrere Verhaltensweisen?
- Navigations-Beispiel:



14

Koordination von Verhalten

➤ Notation:

S: Vektor von Reizen, s_i relevant mit Verhalten β_i im Zeitpunkt t

B: Vektor von aktiven Verhalten β_i im Zeitpunkt t

G: Vektor von Gewinnfaktoren, g_i jedes aktiven Verhaltens

R: Vektor von allen Reaktionen r_i erzeugt von **B**, $\mathbf{B(S)=R}$

Verhaltens-Koordinations-Funktion **C** ist definiert so dass:

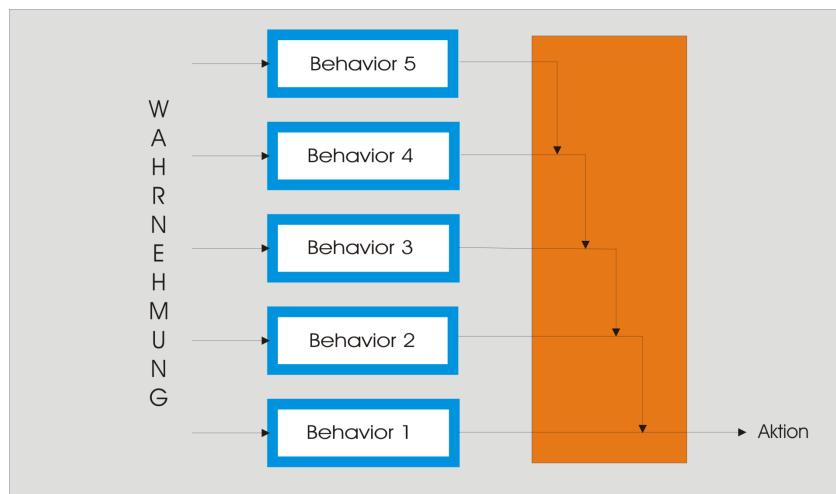
$$\varrho = C(G B(S)) = C(G R)$$

ϱ : Aktion des Roboters in gleicher Form wie r_i

z.B. für Navigation $[x, y, z, \theta, \phi, \psi]$

15

Prioritäten



- Prioritäten bleiben unverändert während der Ausführung;
- Sieht bekannt aus?

16

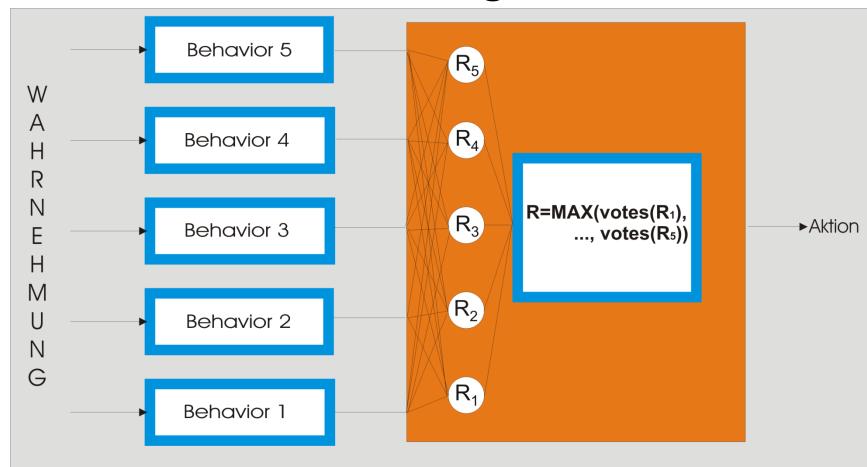
Aktivierungsfunktionen



- Verhalten werden gewählt durch Ziel-Basierte Aktivierungsfunktionen
Prioritäten ändern sich während der Ausführung

17

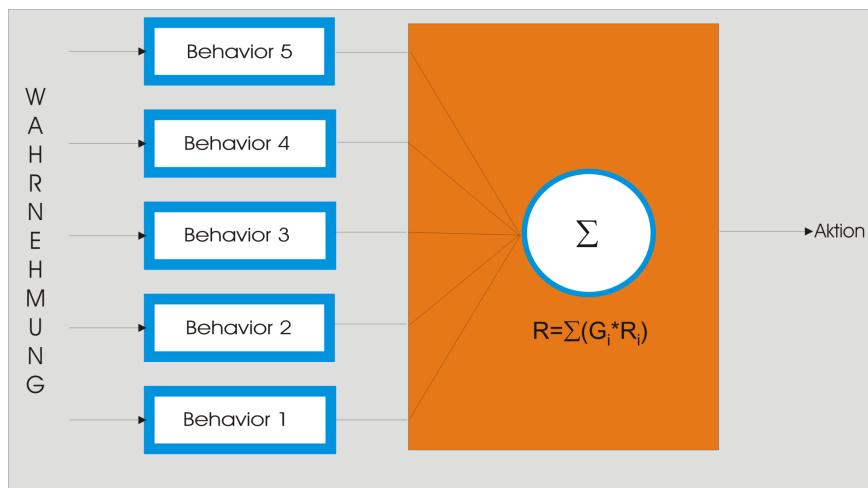
Voting



- Es gibt eine vordefinierte Menge von Reaktionen
 - Jedes Verhalten verteilt Stimmen zu den Reaktionen
 - Die Reaktionen mit den meisten Stimmen werden durchgeführt
- Prioritäten ändern sich während der Ausführung

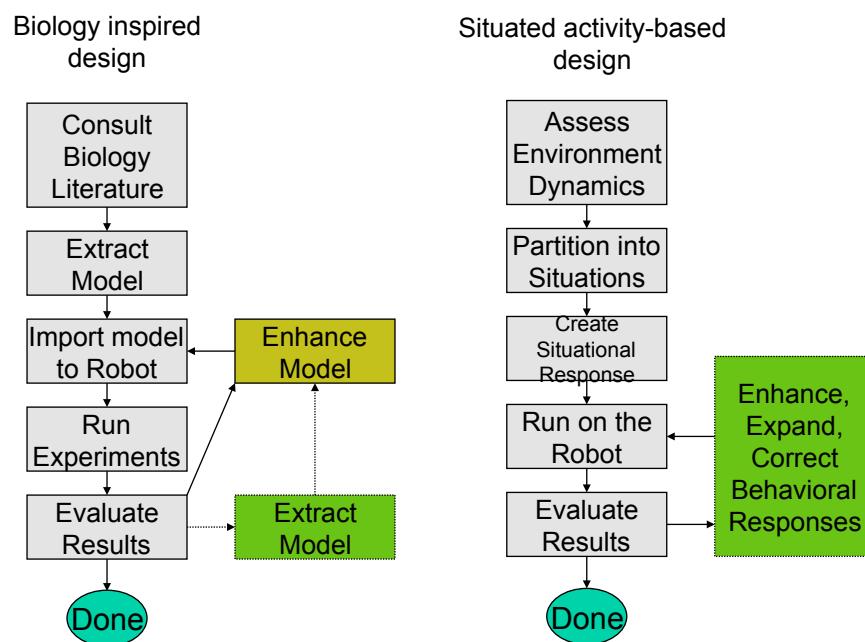
18

Verhaltens-Fusion durch Addition



19

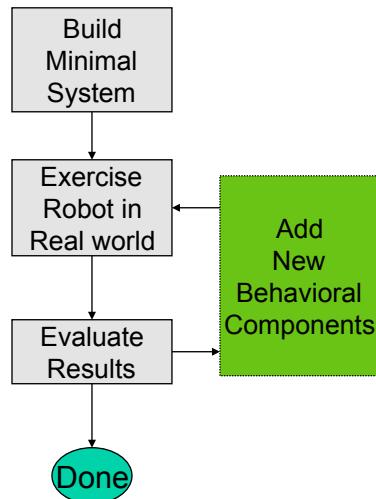
Neue Verhaltensweise entwickeln



20

Neue Verhaltensweise entwickeln

Experimentally driven design



21

Überblick

- Roboter Architekturen
 - Sense-Plan-Act (Deliberative)
 - Reaktive Architekturen
 - Verhaltensorientierte Architekturen
 - **Hybride Architekturen**
- Agenten
 - Rationale Agenten
 - Theorie des Nutzens – Utility Theory
 - Markov Entscheidungsprozesse

22

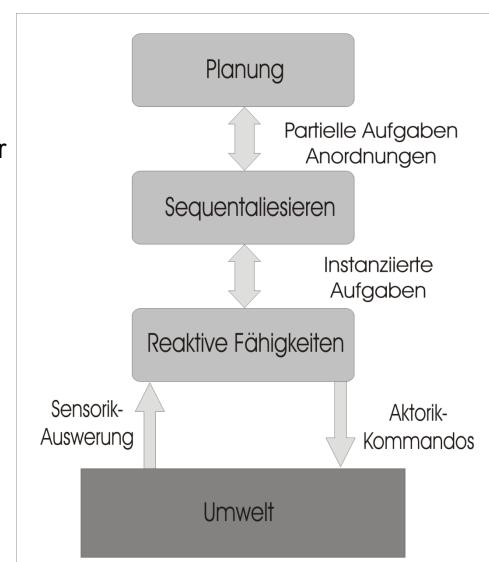
Hybride Architekturen (1)

- Grundidee: „best of the both worlds“ → Kombination von reaktiven und starren (deliberative) Konzepten
- Drei interagierende Ebenen:
 - Eine **Planungskomponente** die sich tiefgehend mit Zielen, Ressourcen und Zeitvorgaben beschäftigt
 - Eine **Ablaufsteuerung**, die eine Menge von Skills aktivieren oder deaktivieren kann, um ein Ablaufnetz zu erzeugen, welches den Zustand der Umgebung verändert und auf diese Art und Weise spezielle Aufgaben löst
 - Eine dynamisch reprogrammierbare Menge von **reaktiven Fähigkeiten** (skills). Sie werden durch den sogenannten Skill Manager koordiniert

23

Hybride Architekturen (2)

- Interaktion der drei Ebenen:
 - Aufgaben in Folgen von Teilaufgaben zerlegen
 - Teilaufgaben sind Folgen von Aktionen oder RAPs (Reactive Action Packages)
 - RAP-Interpreter auf Sequentialisierungs-Ebene zerlegt RAPs in elementare RAPs
 - Ausführen der RAPs durch aktivieren der entsprechenden Fähigkeiten
 - Eine Menge von Ereignis-Monitoren wird ebenfalls aktiviert
 - Sie melden der Sequentialisierungs-Ebene das Auftreten bestimmter Ereignisse in der Umgebung
 - Die aktivierte Fähigkeiten verändern die Umgebung derart, dass die benötigten Ereignisse eintreten



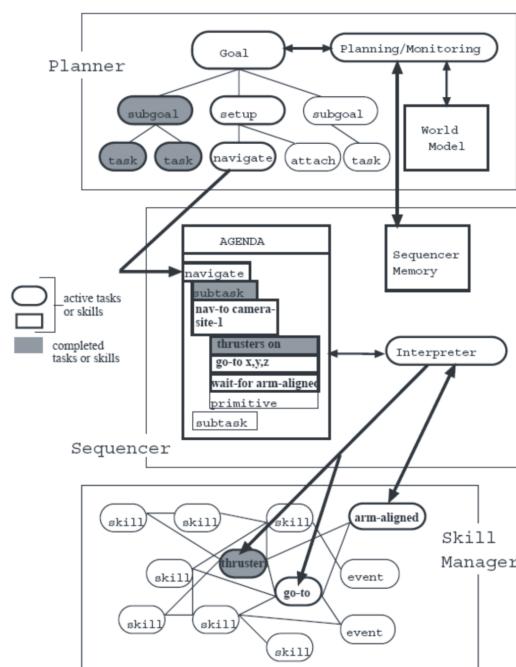
24

Hybride Architekturen (3)

- Die Sequentialisierungs-Ebene beendet die Aktion, wenn
 - Die überwachten Ereignisse eintreten
 - Bestimmte Zeitrahmen überschritten werden
 - Nachrichten aus der Deliberations-Ebene eine Änderung des Plans mitteilen
- Nachteile:
 - stationär
 - konstruiert für bestimmte Hardware-Plattformen und Anwendungen
 - zentrale Überwachung, fehleranfällig
 - keine Fehler-Robustheit
 - keine Erweiterungsmöglichkeit

25

3T NASA



26

Flakey SRI International, '90er



27

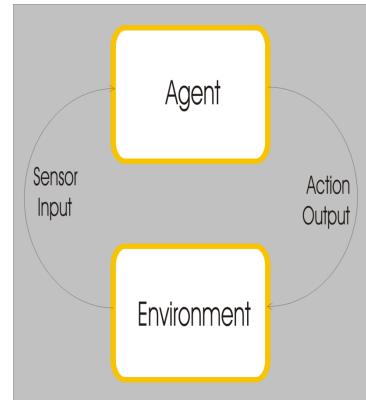
Überblick

- Roboter-Architekturen
 - Sense-Plan-Act (Deliberative)
 - Reaktive Architekturen
 - Verhaltensorientierte Architekturen
 - Hybride Architekturen
- Agenten
 - **Rationale Agenten**
 - Theorie des Nutzens – Utility Theory
 - Markov Entscheidungsprozesse

28

Rationale Agenten

- „An agent is anything that can perceive its environment through sensors and act upon it through actuators“
 - Menschen (Augen → Hände)
 - Roboter (Kameras → Räder)
 - Software-Agenten (GUI)
- Rationale Agenten optimieren bestimmte Leistungsfunktionen
- Sei
 - A: Menge der möglichen Aktionen α
 - O: Menge der Wahrnehmungen o
 - S: Menge der Umwelt-Zustände s
 - π : Strategie (Policy)
- Haupt-Herausforderung: optimale Entscheidung treffen
 - $\pi(o_1, \alpha_1, o_2, \alpha_2, \dots, o_t) = \alpha_t$
 - Markov-Eigenschaft $\pi(s_t) = \alpha_t$
- Stochastischer Übergang zwischen Zuständen
 - „Real-world“ Anwendungen erfordern stochastische Zustandsübergangsfunktionen $P(s_{t+1} | s_t, \alpha_t)$
z.B. Bewegungen von Robotern sind ungenau wegen des Schlupfes der Räder
- Planung durch einfache Graphsuche ist nicht mehr möglich



29

Markov-Eigenschaft

- Mit „Zustand“ zum Zeitpunkt t meinen wir alle Informationen die dem Agent zum Zeitpunkt t über seine Umgebung zur Verfügung stehen
- Der Zustand kann sofortige Wahrnehmungen, weiter verarbeitete Wahrnehmungen und Strukturen, welche über eine Sequenz von Wahrnehmungen aufgebaut wurden beinhalten
- Idealerweise sollte ein Zustand vergangene Wahrnehmungen zusammenfassen, um alle „wichtigen“ Informationen zu erhalten.
Das heißt er sollte die Markov-Eigenschaft besitzen:

$$\Pr\{s_{t+1} = s' | s_t, \alpha_t, \dots, s_0, \alpha_0\} = \Pr\{s_{t+1} = s' | s_t, \alpha_t\}$$

Für alle s' und Zustandsfolgen $s_t, \alpha_t, \dots, s_0, \alpha_0$

30

Überblick

- Roboter-Architekturen
 - Sense-Plan-Act (Deliberative)
 - Reaktive Architekturen
 - Verhaltensorientierte Architekturen
 - Hybride Architekturen
- Agenten
 - Rationale Agenten
 - **Theorie des Nutzens – Utility Theory**
 - Markov Entscheidungsprozesse

31

Theorie des Nutzens – Utility Theory

- Utility wird verwendet, um Zustandspräferenzen von Agenten zu formalisieren
- $\forall s \rightarrow U(s) : U(s) \in \mathbb{R}$
Eine Utility-Funktion ist eine Funktion die einem Zustand (einer Weltbeschreibung) einen metrischen Wert zuordnet - seine Nützlichkeit
- Bedingungen an die Präferenzen oder die Axiome der Theorie des Nutzens:
 - Ordnung: $(A \succ B) \vee (B \succ A) \vee (B \approx A)$
 - Transitivität: $(A \succ B) \vee (B \succ C) \Rightarrow (A \succ C)$
 - Kontinuität: $A \succ B \succ C \Rightarrow \exists [p, A; 1-p, C] \approx B$
 - Substitutivität: $A \approx B \Rightarrow [p, A; 1-p, C] \approx [p, B; 1-p, C]$
 - Monotonie: $A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1-p, B] \approx [p, A; 1-q, B])$
 - Dekomposition: $[p, A; 1-p, [q, B; 1-q, C]] \approx [p, A; (1-p)q, B; (1-p)(1-q), C]$

32

Theorie des Nutzens – Utility Theory

- Utility-Prinzip: Wenn die Präferenzen eines Agenten die Axiome der Theorie des Nutzens erfüllen, dann existiert eine reell-wertige Funktion U, die auf den Zuständen der Welt folgenderweise definiert ist.

$$U(A) > U(B) \Leftrightarrow A \succ B$$

$$U(A) = U(B) \Leftrightarrow A \approx B$$

- Utility zeigt: „Wie glücklich wäre der Agent nach einer Aktion mit dem neuen Umgebungs-Zustand“

33

Entscheiden in einer stochastischen Welt

- Utility-based decision making: Die optimale Aktion α_t^* soll immer das erwartete Utility maximieren

$$\alpha_t^* = \arg \max_{\alpha_t \in A} \sum_{s_{t+1}} P(s_{t+1} | s_t, \alpha_t) U(s_{t+1})$$

- Da jedem Umwelt-Zustand ein Utility-Wert zugeordnet ist, kann eine optimale Aktion für jeden Zustand berechnet werden
→ Eine „greedy“ (gierige) Strategie ist auch optimal

$$\pi^*(s) = \arg \max_{\alpha} \sum_{s'} P(s' | s, \alpha) U^*(s')$$

- Wir haben eine optimale aber statische Strategie kalkuliert
Wie entscheidet ein Agent für fortlaufende Aktionen?

34

Überblick

- Roboter-Architekturen
 - Sense-Plan-Act (Deliberative)
 - Reaktive Architekturen
 - Verhaltensorientierte Architekturen
 - Hybride Architekturen
- Agenten
 - Rationale Agenten
 - Theorie des Nutzens – Utility Theory
 - **Markov Entscheidungsprozesse**

35

Markov Entscheidungsprozesse (1)

- Ein Markov Entscheidungsprozess (MDP) ist ein Tupel $\langle S, A, P, R \rangle$
 - S: Menge der Umwelt-Zustände s
 - Zustandsübergangsfunktion $P = \Pr(s' | s, \alpha)$
 - A: Menge der möglichen Aktionen α
 - Belohnungsfunktion $R(s, \alpha) = E\{r_{t+1} | s_t = s, \alpha_t = \alpha, s_{t+1} = s'\} \quad \forall s, s' \in S, \alpha \in A$
- Die Utility-Funktion eines Zustandes ist die erwartete Belohnung, beginnend ab diesem Zustand; hängt von der Strategie des Agenten ab:

$$U(s) := R(s, \alpha) + \gamma \sum_{s'} P(s' | s, \alpha) U(s')$$

Utility-Function Sofortige Belohnung Zukünftige Belohnung

36

Markov Entscheidungsprozesse (2)

- Die optimale Strategie ist definiert (Bellman-Gleichung):

$$\pi_t^*(s) = \arg \max_{\alpha} \{ R(s, \alpha) + \gamma \sum_{s'} P(s' | s, \alpha) U_{t-1}^*(s') \}$$

- Um eine optimale Strategie über die Lösung der optimalen Bellman-Gleichung zu finden benötigt man folgendes:
 - Exakte Kenntnis der Dynamik der Umgebung
 - Genug Speicherplatz und Berechnungszeit
 - Die Markov-Eigenschaft
- Wieviel Speicherplatz und Zeit?
 - polynomiell mit der Anzahl der Zustände (über Dynamic Programming Methoden)
 - ABER, normalerweise ist die Anzahl der Zustände groß
- Man muss auf Approximationen zurückgreifen

37

MDP Beispiel

Recycling Roboter

- Zu jedem Zeitpunkt muss der Roboter entscheiden, ob er
 - aktiv eine Dose suchen soll
 - auf jemanden warten soll, der ihm eine Dose bringt
 - zum Aufladen der Batterie zur Basisstation fahren soll
- Suchen ist besser, verbraucht allerdings Energie; wenn die Batterien leer laufen während er sucht, muss er „befreit“ werden (schlecht)
- Entscheidungen werden auf Grundlage des momentanen Energilevels getroffen: high, low
- Belohnung = Anzahl an gesammelten Dosen

38

Recycling Roboter MDP

$$S = \{\text{high}, \text{low}\}$$

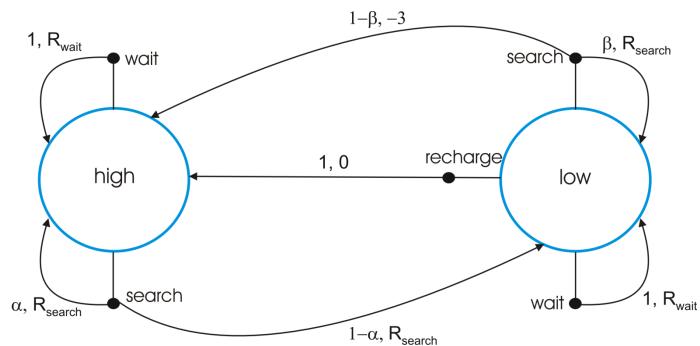
$$A(\text{high}) = \{\text{search}, \text{wait}\}$$

$$A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$$

R_{search} = erwartete Anzahl an Dosen während der Suche

R_{wait} = erwartete Anzahl an Dosen während des Wartens

$$R_{\text{search}} > R_{\text{wait}}$$



39