2.5. Kontrollstruktur

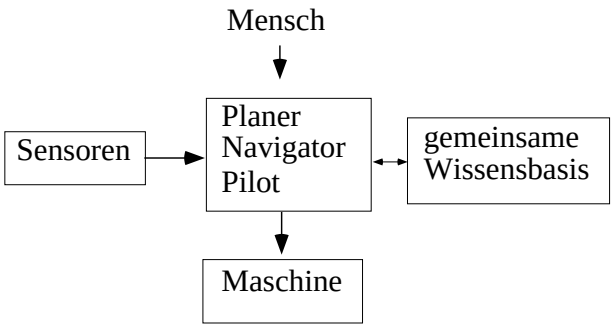
Architektur der Datenverarbeitung für den AMR

Aufgaben:

- Sensordatenvorverarbeitung
- Fusion der Sensordaten in logischen Sensoren für
  - Lokalisation
  - Erstellung einer Hinderniskarte
  - Erstellung von Karten für Navigation und Aktionsplanung
  - Erkennen von Objekten
  - Darstellung und Repräsentation von Wissen
- Motorregelung: Realisierung einer (vω) - Schnittstelle
- Pilotfunktion: Bahnregelung über Vorgabe von v und ω, verantwortlich für
  - Kollisionsvermeidung
  - Manövrieren in engen Passagen
  - Wenden in Sackgassen
- Navigation: ermittelt befahrbare Bahnen für die Pilotfunktion; plant Umgehung von Hindernissen, kennt Umgebung
- Planungsfunktion: Planung von Aktionen und Vorgaben an den Navigator; trifft strategische Entscheidungen, hat weiten Überblick
- Benutzerkontakt: Mensch-Maschine-Schnittstelle (MMI)

2.5.1. Horizontale (hierarchische) Kontrollstruktur

Konzentration auf die Aktionen im AMR



drei Hierarchieebenen

- Planer
- Navigator
- Pilot

Planer: hat weitesten Überblick über Umgebung, kennt nur die wichtigsten Details

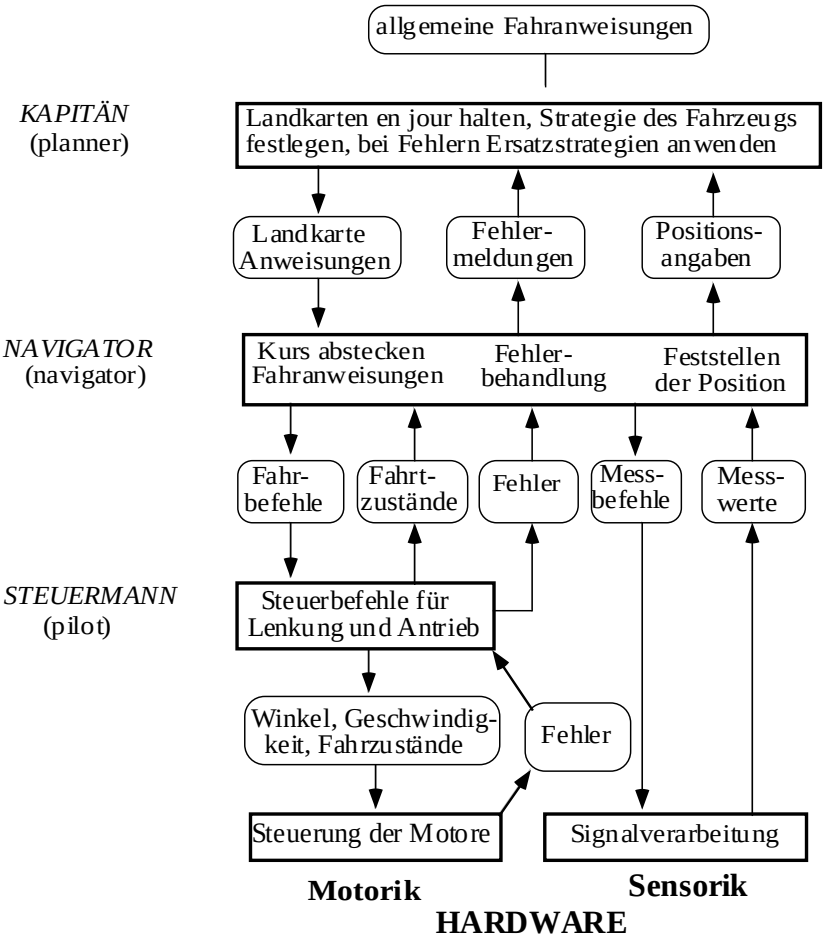
Navigator: kennt das Fahrzeug und die Umgebung bis zum nächsten Teilziel

Pilot: kennt Position des AMR und unmittelbare Umgebung

2.5.1.1. Funktionale Gliederung der Steuerung des Robots

Der allererste Autonome Mobile Roboter, Shakey, von Nilson 1969 am Stanford Research Institute (SRI) gebaut, hatte diese Kontrollstruktur.

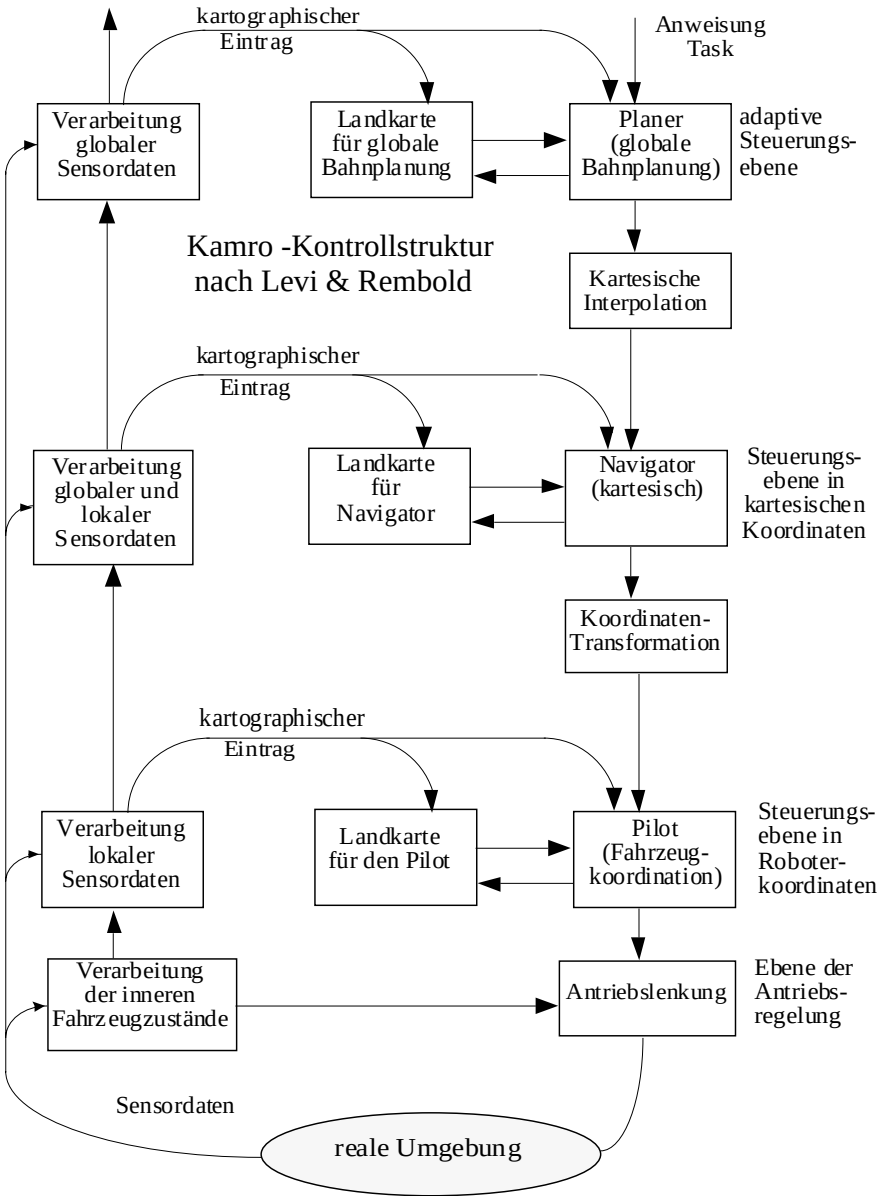
(Nilson 1969)



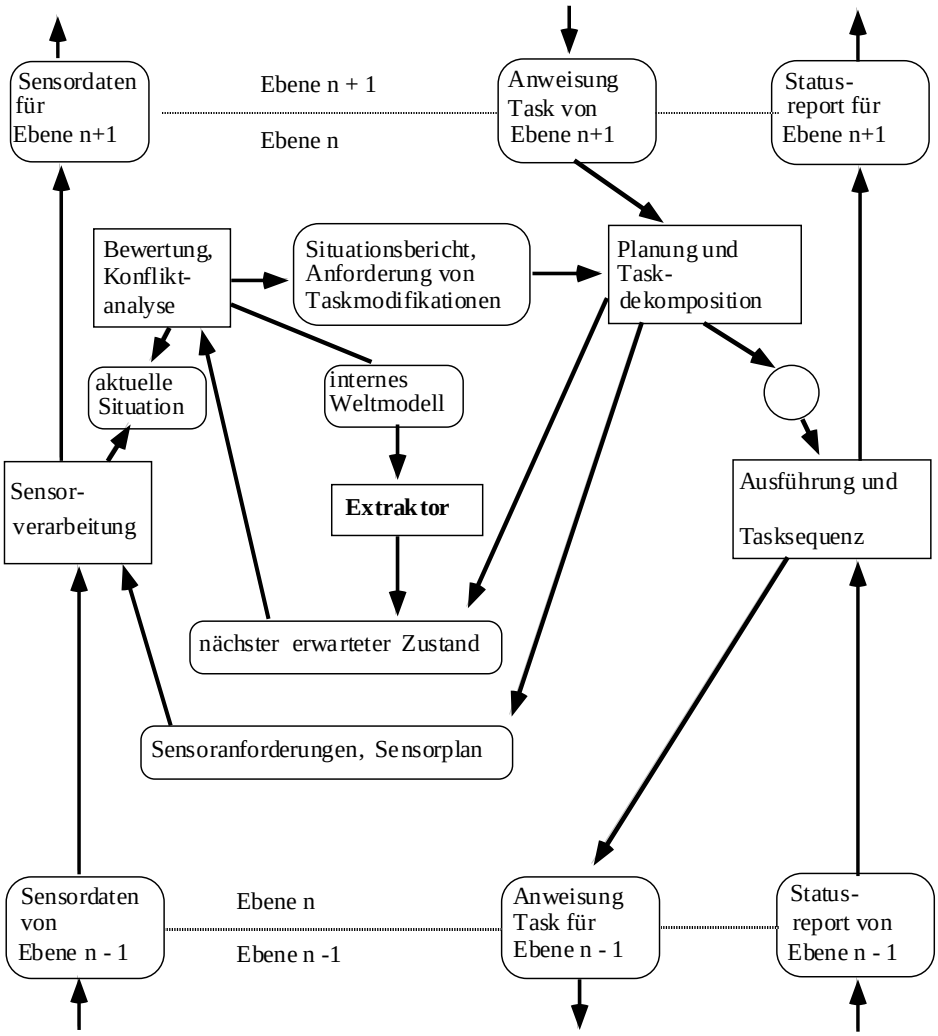
Diese Art von Kontrollstruktur ist zunächst naheliegend und beschreibt eine natürliche Aufgabenteilung in der Datenverarbeitung eines AMR. Sie hat allerdings den Nachteil, dass der AMR nur dann korrekt arbeitet, wenn alle Teile funktionieren.

2.5.1.2. Schichtenstruktur des KAMRO

Der Karlsruher Mobile Roboter von Rembold und Levi hat eine ausgeprägte Schichtenstruktur der Software.



Eine DV-Schicht verarbeitet lokal einen Teil der Sensordaten und reicht den anderen Teil in die nächst höhere Schicht weiter. Sie erzeugt eine schichtspezifische Repräsentation der Umwelt und leitet daraus die Steuerbefehle dieser Schicht ab. Ein Teil der Aufgabe wird dabei an die nächst tiefere Schicht delegiert. Jede Schicht erzeugt Rückmeldungen an die nächst höhere Schicht.



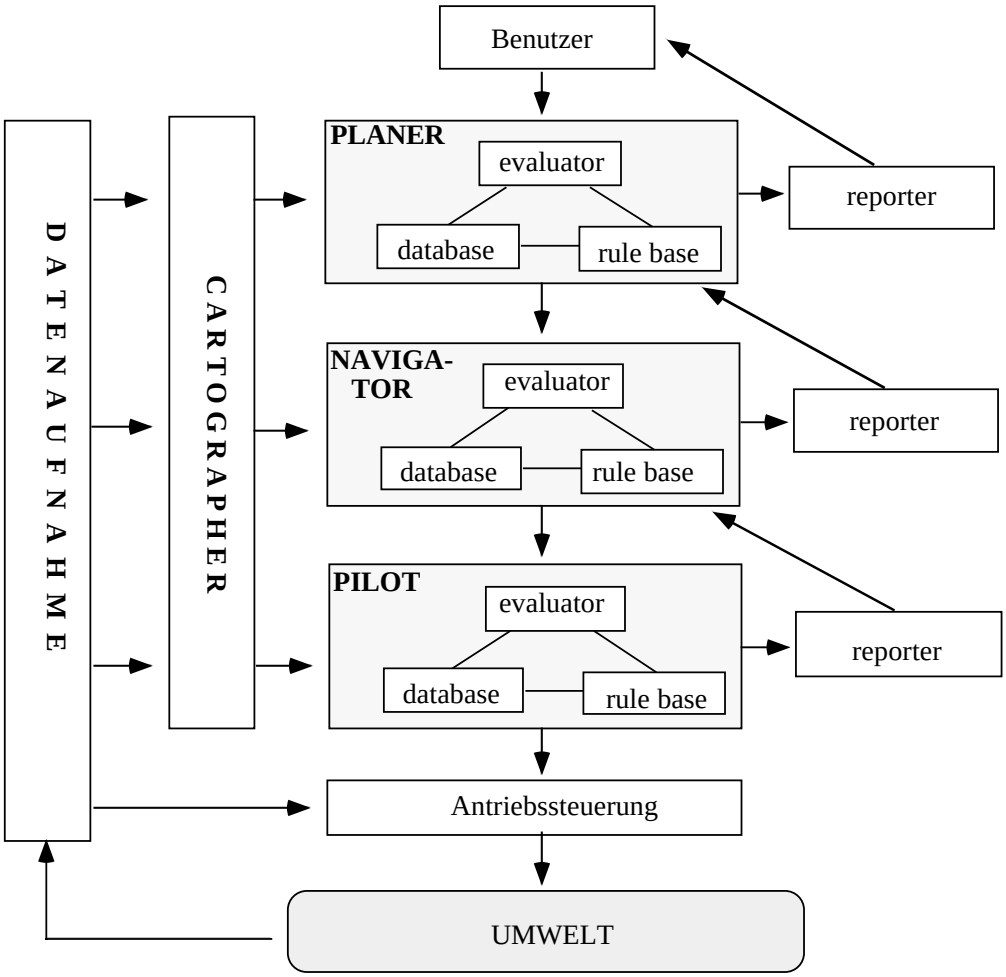
2.5.1.3. Hierarchische Kontrollstruktur nach Isik & Meystel

Die einzelnen Ebenen, Planer, Navigator und Pilot, haben je eine eigene Datenbasis und ein Regelsystem, das ihre Aktionen steuert. Sie sind verbunden über einen gemeinsamen Kartographer, der die Daten der Sensoren zu Karten der Umwelt aufbereitet und den Ebenen zur Verfügung stellt.

Kommandos gehen von oben nach unten zu der Antriebseinheit, Rückmeldungen werden von "reportern" erzeugt. Diese Rückmeldungen sind:

Pilot --> Navigator

- Abweichungen der Bahn von der Vorgabe
  (Sackgassen, nicht umfahrbare Hindernisse, zu weites Abdriften vom Sollkurs)
- Abweichungen der Fahrgeschwindigkeit vom Soll (Notbremsungen)
- Betriebsstörungen, die der Pilot nicht abfangen kann
- Anforderung von Teilzielen / Bahnsegmenten nach Erreichen des letzten Teilziels



Rückmeldungen Navigator --> Planer

- Erreichen von Teilzielen und Anforderung neuer Teilziele
- Behinderungen (Wegblockaden)
- vom Navigator nicht abfangbare Betriebsstörungen

Rückmeldungen Planer --> Benutzer

- Fertigmeldungen (Endziel ereicht), Bereitmeldungen
- vom Planer nicht abfangbare Betriebsstörungen

Die Wissensbasis jeder Ebene hat die Aufgaben

- Vergleich: bekanntes Wissen <--> Umwelteindrücke
- Zuordnung: Objekte <--> Bedeutung (Hindernisse / Freiraum)
- Verdichtung: Wissen --> Repräsentationen der Umwelt

Der Kartographer versorgt die hierarchischen Ebenen mit Repräsentationen der Umwelt

- Pilot: Hindernisinformation / -Karte (robotzentriert)
- Navigator: geometrische Karte der Umgebung (in Weltkoordinaten)
- Planer: Graph der Gesamtumgebung

Die einzelnen Ebenen haben unterschiedliche Zeitrahmen

- Pilot: ~ 0.1 s
  Steuerung des AMR in Echtzeit
  Reaktion auf Fahrbefehle
- Navigator: ~ s
  erzeugt Fahrbefehle zum Anfahren von Wegpunkten oder gibt Bahnen vor
  hat Zeit bis der AMR den nächsten Wegpunkt / Bahnabschnitt erreicht hat
- Planer: weiter Zeithorizont
  theoretisch nur einmal tätig; komplexe Algorithmen zur Wegeplanung

Nachteile der hierarchischen Kontrollstruktur:

- der AMR funktioniert nur, wenn **alle** Komponenten funktionieren
- das Verhalten kann nur schwer von außen überprüft werden

Verbesserungen:

- funktionale Schnittstellen
- testbar über die zwischen Funktionen ausgetauschten Daten

2.5.2. Vertikale (verhaltensorientierte) Kontrollstruktur Rodney A. Brooks, MIT, 1985/86

2.5.2.1. Forderungen an einen AMR

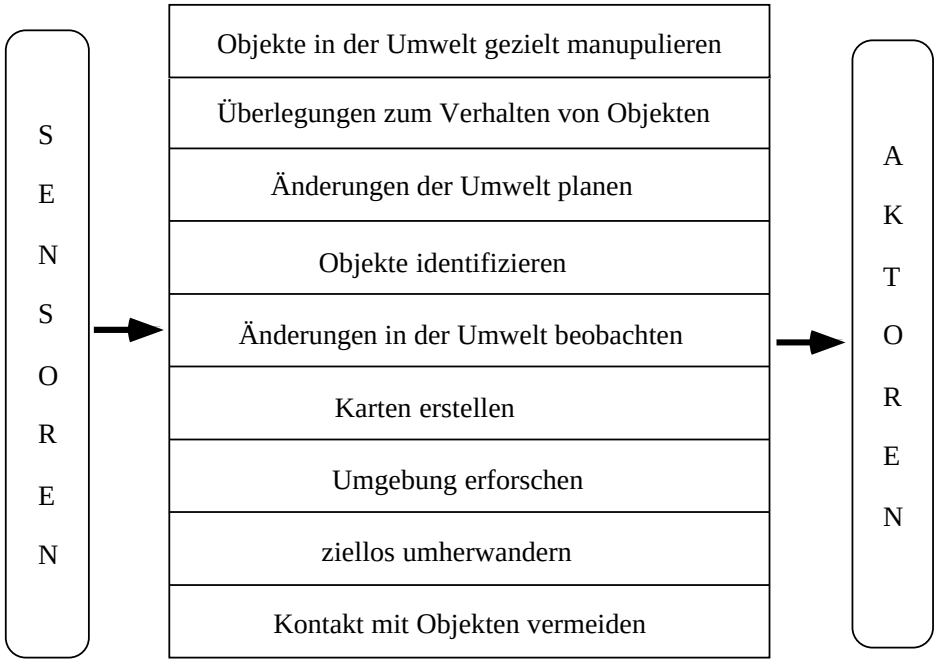
Fehlertoleranz	gegenüber Ausfällen von Hard- und Softwarekomponenten	
Erweiterbarkeit	ohne grundlegende Änderung des bestehenden Gesamtsystems	
Testbarkeit	in jedem Entwicklungsstadium	multiple goals

Umgang mit konkurrierenden Zielen und Konfliktlösung

==> **verhaltensorientierte Struktur:** Funktionsebenen --> Verhaltensebenen

- jede Schicht repräsentiert und realisiert ein Verhalten und verfolgt ein Ziel
- jede Schicht arbeitet autonom
- jedes Verhalten einer höheren Schicht enthält das Verhalten aller tieferen Schichten
- jede tiefere Schicht ist von höheren unabhängig
  ==> Ausfall des Gesamtsystems erst bei Ausfall der untersten Schicht

2.5.2.2. Verhaltensorientierte Kontrollstruktur nach Brooks



2.5.2.2.1. Rechnersystem zur Realisierung der Schichtenstruktur

Lose gekoppeltes Netz aus möglichst einfachen Prozessoren und Softwaremoduln

- jede Schicht besteht aus einem Netzwerk mehrerer Module (mehrere Prozessoren)
- die Prozessoren arbeiten asynchron
- Module können nicht direkt auf Daten anderer Module zugreifen
  (man braucht keinen gemeinsamen Speicher für alle Prozessoren)
- die Module kommunizieren unidirektional durch Versenden von Nachrichten
- Nachrichten werden nicht quittiert, dürfen verfälscht ankommen oder verloren gehen ohne die Funktion des Gesamtsystems zu beeinträchtigen
- mit der Realisierung der untersten Verhaltensschicht
  "Kontakt mit Objekten vermeiden" beginnt das System zu fahren (auszuweichen)

2.5.2.2.2. Module

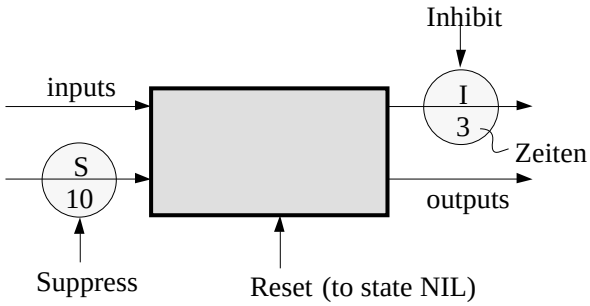
Definition

Ein Modul ist eine Berechnungseinheit, die

- Nachrichten (Daten) erhält
- diese verarbeitet
- Nachrichten an andere Module versendet

Beeinflussung von außen durch …

- Unterdrücken der Eingangswerte (suppress)
  und Ersetzen durch die des unterdrückenden Moduls
  während der angegebenen Zeit (in Sekunden)
- Unterdrücken der Ausgangswerte (inhibit) während der angegebenen Zeit



2.5.2.2.3. Struktur eines Moduls

endlicher Automat (finite state machine)

definierte Zustände (states)

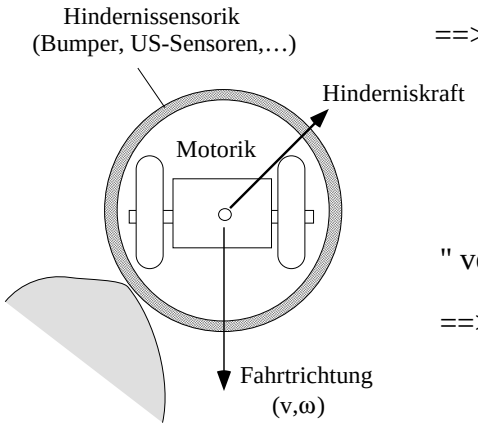
festgelegte Operationen:

- **Output**
  Funktion (Eingänge, interne Variable) ---> Ausgangsnachricht ---> Folgezustand
- **Side effect**
  Funktion (Eingänge, interne Variable) ---> neue interne Variable ---> Folgezustand
- **Conditional Dispatch**
  Prädikat (Eingänge, interne Variable) ---> Verzweigung in einen Folgezustand
- **Event Dispatch**
  - alle Eingänge werden auf das Eintreffen von Nachrichten abgefragt
  - alle Bedingungen werden nacheinander berechnet
  - das System wartet, bis eine der Bedingungen erfüllt ist ----> Folgezustand

Im Folgenden wird zunächst als Hardware ein AMR angenommen mit

- einem Differentialantrieb
- einer Hindernissensorik in Form einer Bumperschürze oder Ultraschallsensoren
- einem logischen Kontaktsensor
  (fühlt eine Hinderniskraft, entgegengesetzt einem Kontaktpunkt)
- einer Stereokamera, mit nachgeschalteter Bildauswertung

Als Beispiele für Module werden betrachtet ein Modul "runaway" und ein weiterer Modul "avoid".



==> log. Sensor "feel force"

Vektor vom Hindernis fortzeigend

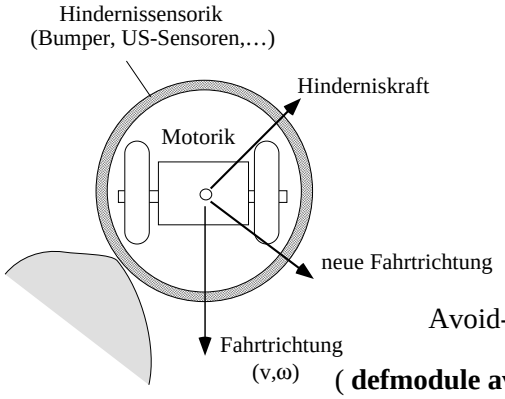
" vermeide Kontakt mit Objekten"

==> Fahre in Richtung "Hinderniskraft"

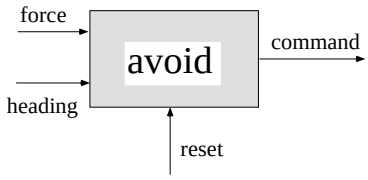
bis die Kraft ≈ Null ist

falls durch eigene Bewegung Kollision droht

halte an



Avoid-Modul



Avoid-Modul Beschreibung in LISP

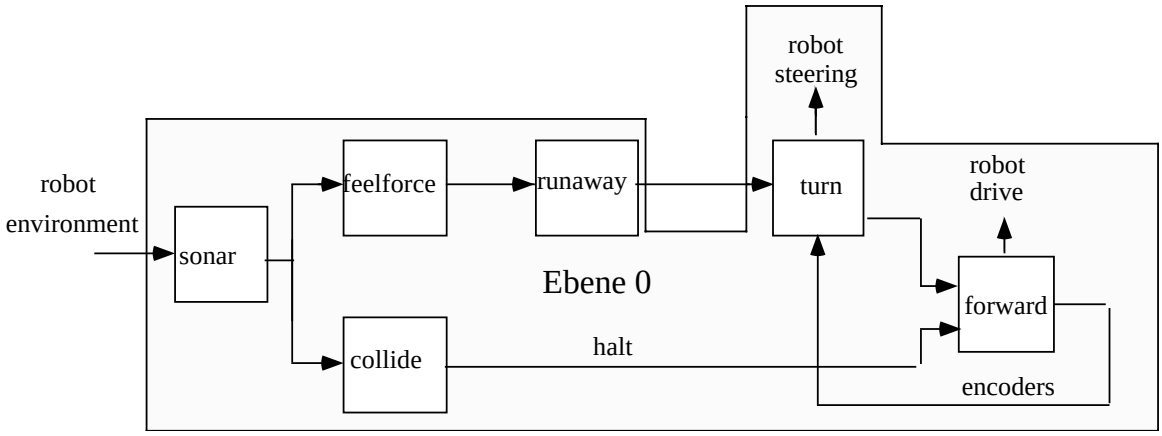
```
( defmodule avoid-1
        : inputs          (force heading)
        : outputs         (command)
        : instance-vars   (resultforce)
        : states

  ( ( NIL ( event-dispatch ( and force heading ) plan )
    ( plan ( setf resultforce ( select-direction force heading ) go ))
    ( go    ( conditional-dispatch ( significant-force-p resultforce
                                                    1.0 ) start NIL))
    ( start  ( output-command ( follow-force resultforce) NIL     ))
  )
 )
```

2.5.2.2.4. Verhalten der Ebene 0

Ziel: Kontakt mit Objekten vermeiden

Taktik:
- Nähert sich ein Objekt, so laufe in entgegengesetzter Richtung davon
- Falls Kollision durch eigene Bewegung droht, halte an



Module der Ebene 0

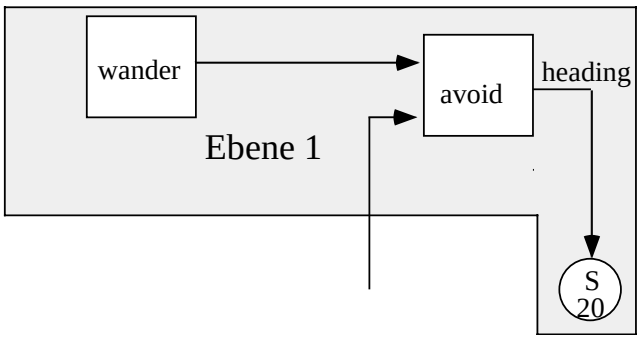
- *sonar* nimmt einen Vektor von Sonardaten auf, filtert die Daten und erstellt robotzentrierte Karte in Polarkoordinaten
- *collide* erzeugt HALT-Nachricht bei Kontakt mit Hindernissen
- *feelforce* summiert die Sensorsignale der entdeckten Objekte auf und berechnet daraus eine abstoßende Gesamtkraft
- *turn* erzeugt aus dem Richtungsbefehl die Drehung des AMR und einen Fahrbefehl (erzeugt BUSY-Signal, solange gedreht wird)
- *forward* erzeugt den Fahrbefehl der Motore, solange keine HALT-Nachricht vorliegt

2.5.2.2.5. Verhalten der Ebene 1

Ziel: ziellos umherwandern

Module der Ebene 1

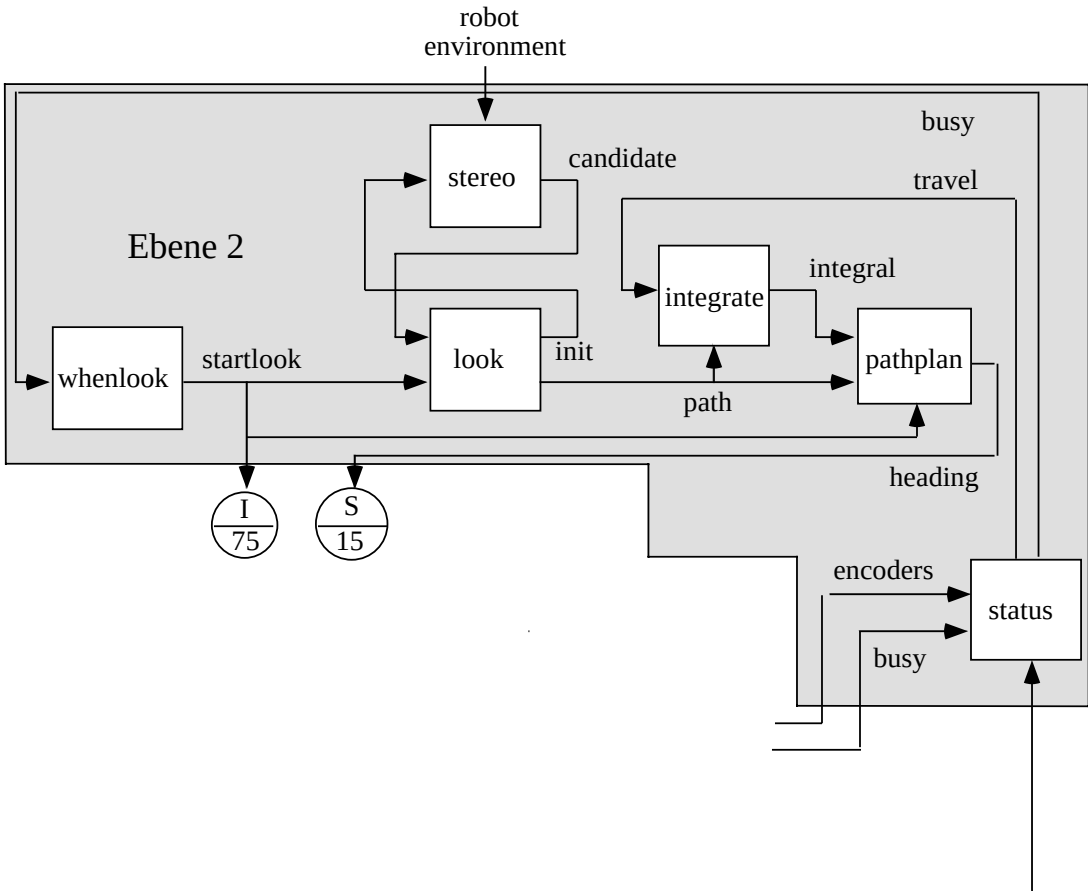
- *wander* erzeugt durch Zufallsgenerator alle 10 s neuen Fahrbefehl
- *avoid* verrechnet Fahrbefehl mit abstoßender Gesamtkraft zu neuem Richtungs-fahrbefehl, der 20 s die Befehle von RUNAWAY unterdrückt



2.5.2.2.6. Verhalten der Ebene 2

Ziel: Umgebung erforschen

==> Erweiterung der Ebene 1



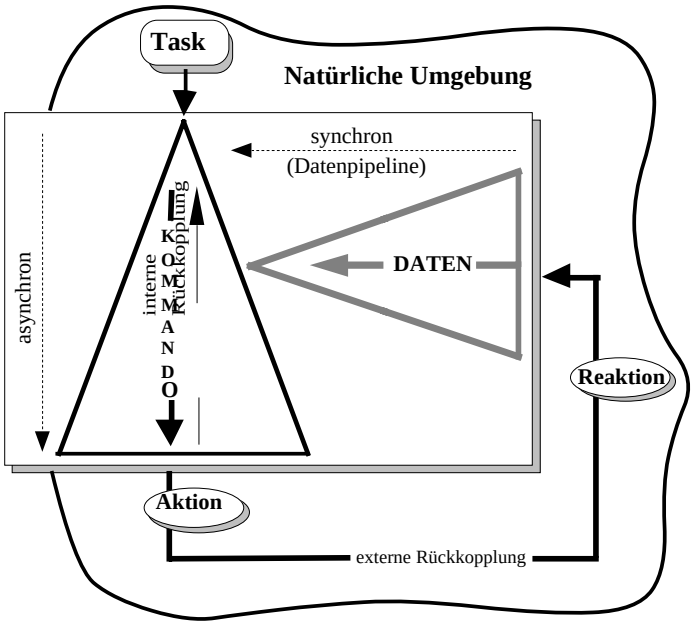
Module der Ebene 2

- *whenlook* blockiert bei Anhalten des AMR den Zufallsgenerator und fordert Aufnahme eines Bildes an (STARTLOOK)
- *look* initiiert eine stereo-Aufnahme
- *stereo* findet freien Korridor, in dem der AMR fahren kann
- *integrate* integriert die Signale der internen Navigationssensoren zu einem Weg und der Position des AMR
- *pathplan* korreliert die Korridor- und Positionsangabe zu einem Richtungsbefehl, der in AVOID eingespeist wird

Die verhaltensorientierte Kontrollstruktur modelliert den AMR als reaktives System. Sie tut sich schwer mit der Modellierung von höheren Planungsebenen.

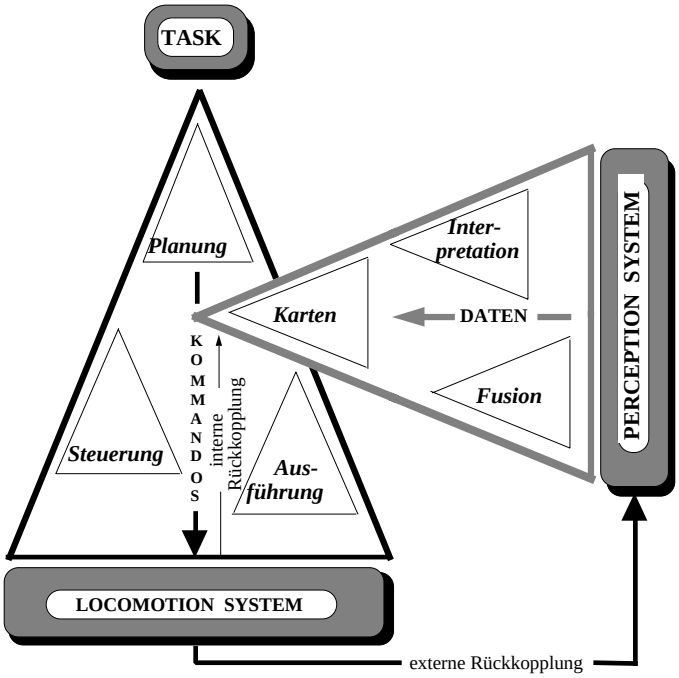
2.5.3. Orthogonale Kontrollstruktur

Sie beschreibt die Softwarestruktur als orthogonales System von Datenerfassung und –verdichtung und Kommandoerzeugung für die Antriebseinheiten des AMR.



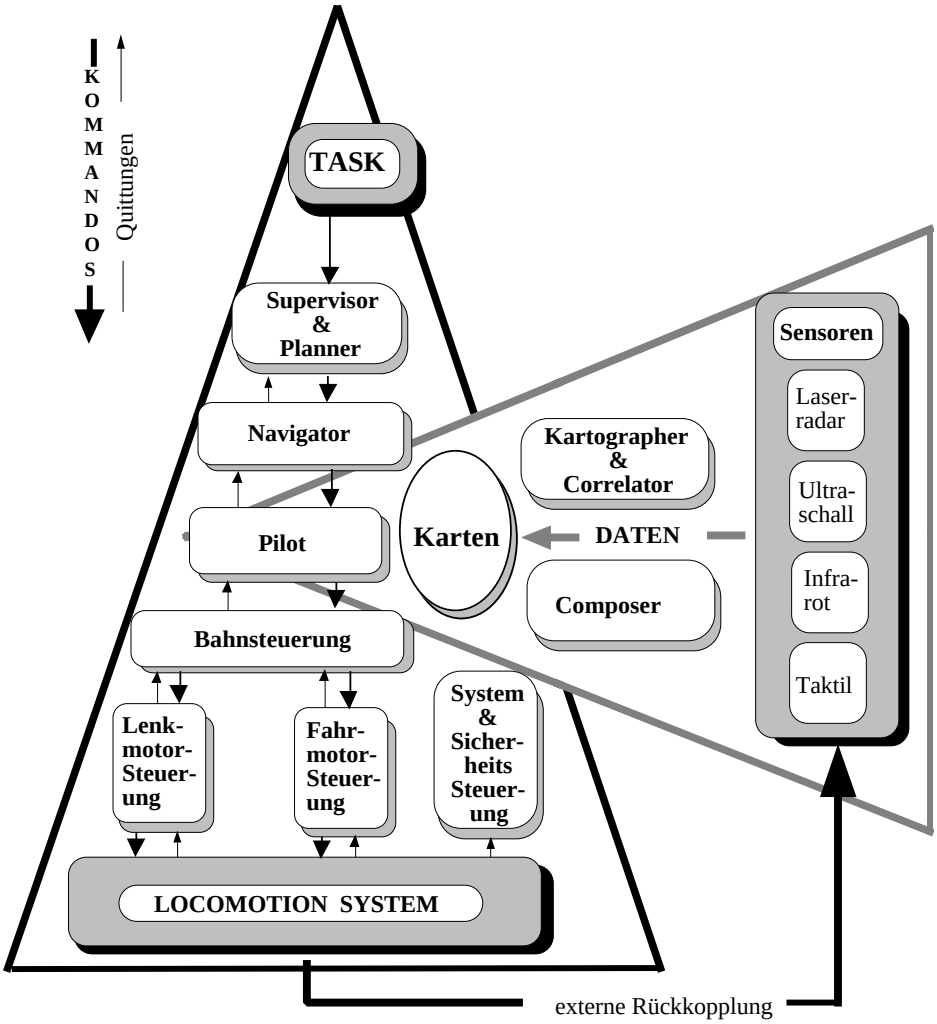
Die Datenerfassung und -verdichtung teilt sich ihrerseits auf in

- Fusion der Daten zu einem in sich konsistenten Bild der Umwelt
- Interpretation der Daten für die Zwecke des Robots
- Erzeugung von "Karten" für die ausführenden Komponenten



Der die Kommandos generierende Ast der DV benutzt die "Karten" für Module "Planung", "Steuerung" und "Ausführung", die über Kommandos und Rückmeldungen miteinander kommunizieren. Von außen wird dem System eine Reihe von Tasks vorgegeben und über die Umwelt geschieht eine Rückkopplung zwischen dem Bewegungsapparat und der Sensorik des AMR.

Eine feinere Aufteilung der Aufgaben zeigt das nächste Bild:



Waagerechter Ast - Sensordatenverarbeitung

- *Composer* — setzt aus den verschiedenen Sensorquellen ein Signal zusammen
- *Korrelator* — korreliert eine Momentaufnahme mit vorhandenen Bildern, um Mehrfachaufnahmen desselben Objekts zu vermeiden
- *Karthographer* — erzeugt aktuelle Karten für Planer, Navigator und Pilot
- *System & Sicherheitssteuerung* — überwacht Ströme, Temperaturen, Spannungen, …

Senkrechter Ast - Kommandoerzeugung

- *Supervisor & Planer*	plant graphenorientierte Wege und überwacht ihre Ausführung
- *Navigator*	plant vom AMR befahrbare Wege als Folge von Kursvektoren und überwacht ihre Ausführung im Piloten
- *Pilot*	berechnet Bahnsegmente, setzt sie um in (v, ω)-Kommandos unter Berücksichtigung der aktuellen Hindernissituation meldet abgefahrene Kursvektoren zurück
- *Bahnsteuerung*	überwacht die Einhaltung der vorgegebenen Bahn, regelt Abweichungen aus.