

# Kapitel 7 Navigation

1. Zum Einstieg: Worum geht es?
2. Sensorik
3. Sensordatenverarbeitung
4. Fortbewegung
5. Lokalisierung in K
6. Kartierung
7. **Navigation**
8. Umgebungsdaten
9. Roboterkontrollarchitekturen

## **7.1 Hintergrund**

## **7.2 Reflex-basierte Navigation**

## **7.3 Reaktive Navigation**

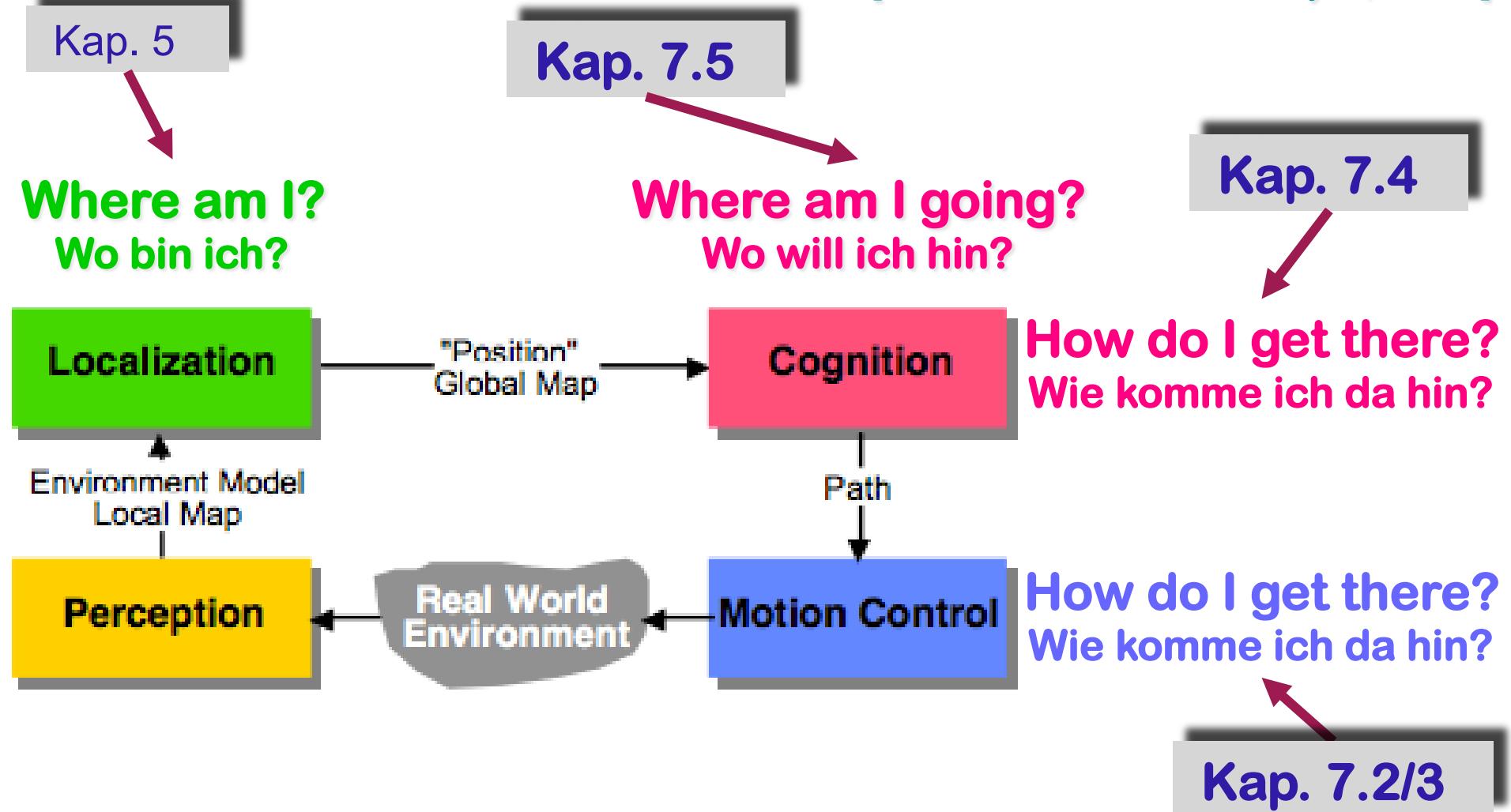
## **7.4 Pfadplanung**

## **7.5 Explorationsplanung**

Ausblick

# 7.1 Hintergrund

nach [Leonard & Durrant-Whyte, 1991]



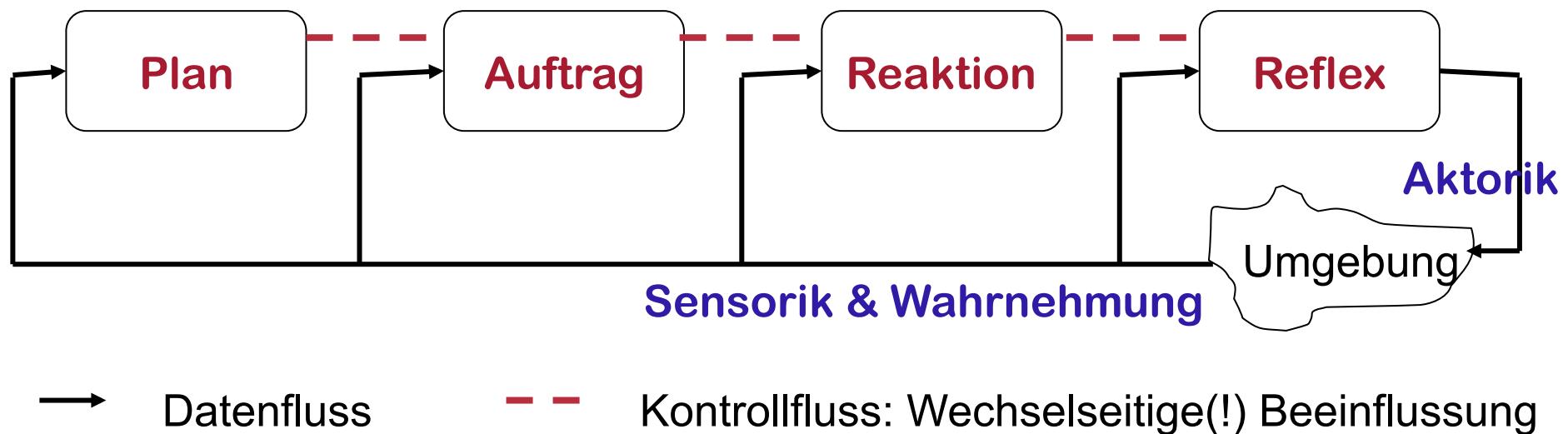
# Modulkette zur Roboterkontrolle

## Vorgriff auf Kapitel 9

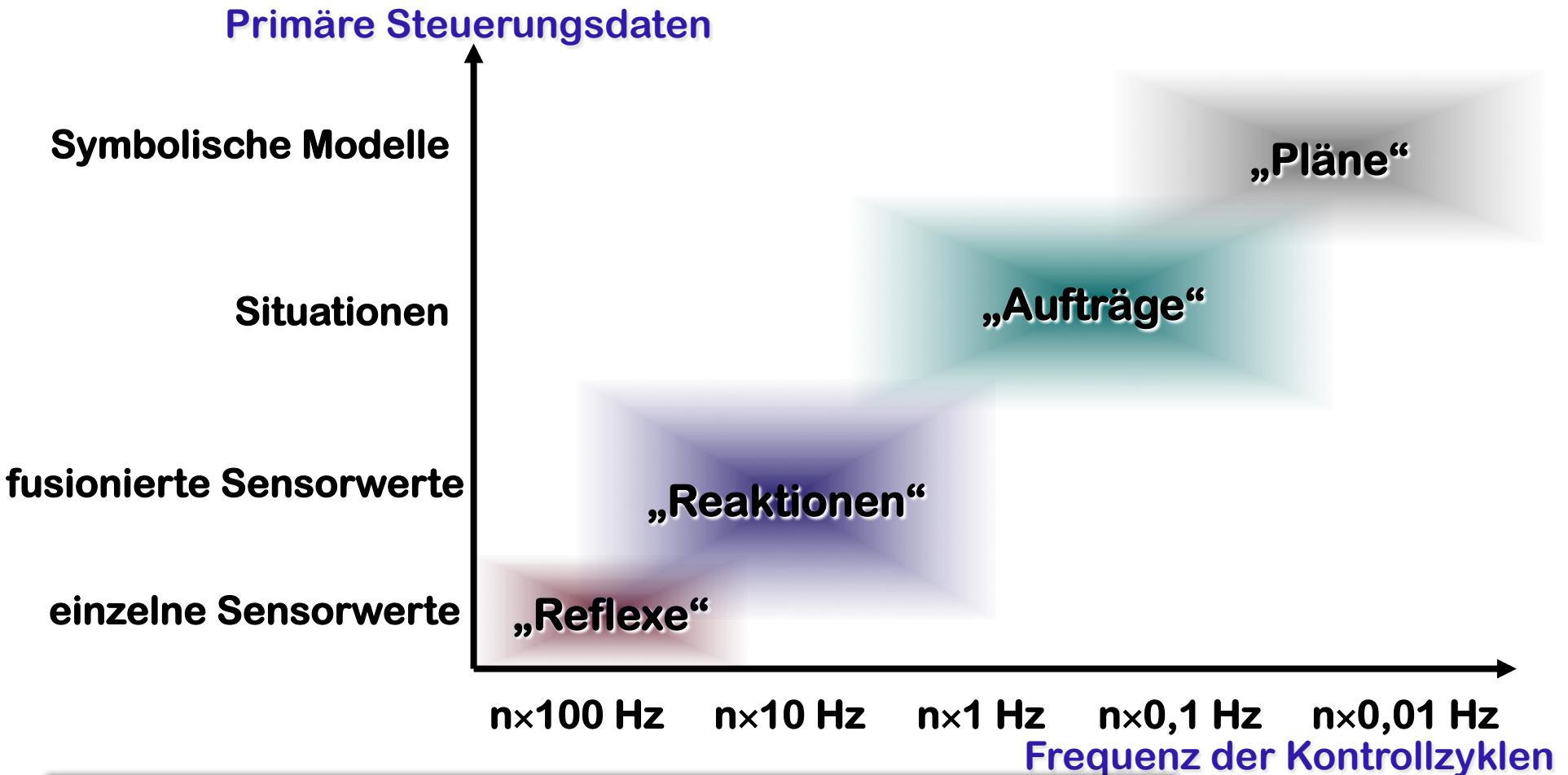
Zur Navigation (nicht nur dafür) ist starre Folge inadäquat!

Sensormessen → Interpretieren → Schließen → Handeln

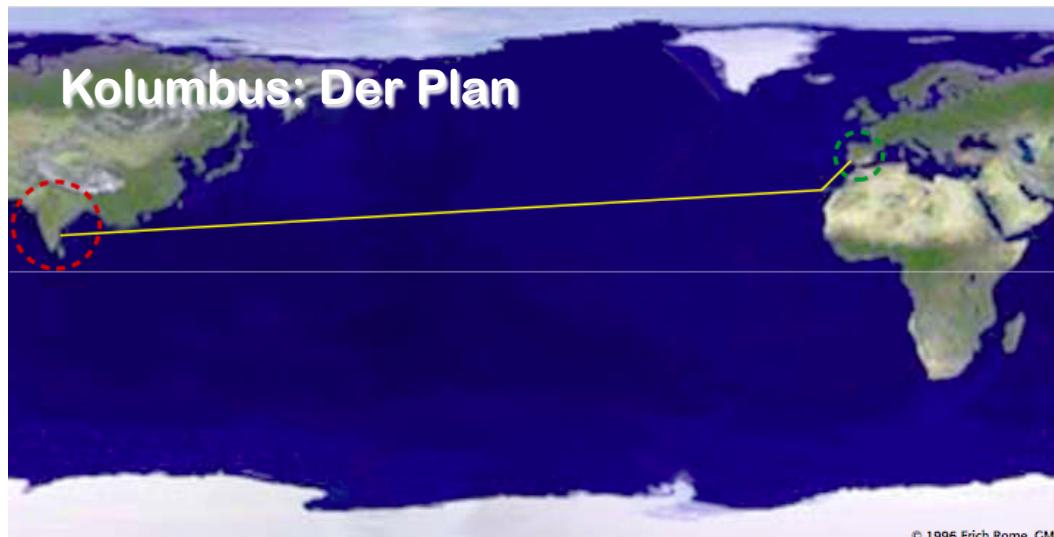
Grundmuster stattdessen: **Modulkette**



# Zeitskalen und Granularitäten



# Pläne sind wichtig ...



... aber manchmal läuft es einfach anders!

Wie verzahne ich  
Planung und Reaktion etc.  
im wirklichen (Roboter-)Leben?

Kap. 9



# The World is its own Best Model!

- **Verhaltensbasierte** (*behavior-based*) KI/Robotik entstand etwa Mitte der 1980er Jahre (R. Brooks u.v.a.)
- Sie wandte sich zu Recht gegen Überbetonung symbolorientierter Steuerungsebenen für KI-Systeme
  - Steuere auf allen Ebenen nach Sensordaten, nicht nach Umg.-Modell!
- Teilweise ersetzte sie diese durch Überbetonung der nicht-symbolorientierten Steuerungsebenen
- Entspannte Sicht von heute: Man sollte beide gebrauchen

## 7.2 Reflex-basierte Navigation

### Unterschied Reflex/Reaktion

- **Reflexe** basieren auf elementaren Sensordaten; überschreiben andere Steuerungsbefehle (hart!)
  - Rückzugsreflex bei Laufmaschine SCORPION (wenn Bein anstößt, zieh es zurück & schwinge höher)
- **Reaktionen** basieren auf (mild) fusionierten Sensordaten; modifizieren andere Steuerungsbefehle
  - „Wichte“ Kollisionsvermeidung mit Richtung des Fahrziels
  - Schalte Kollisionsvermeidung selektiv aus für Roboterfußball



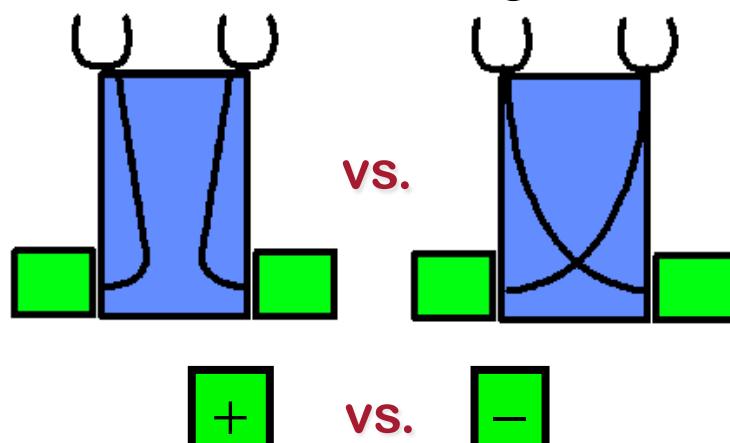
# Braitenberg'sche Vehikel

V. Braitenberg: Vehicles: Experiments in Synthetic Psychology.  
MIT Press, 1984. Deutsch: „Künstliche Wesen“, 1986

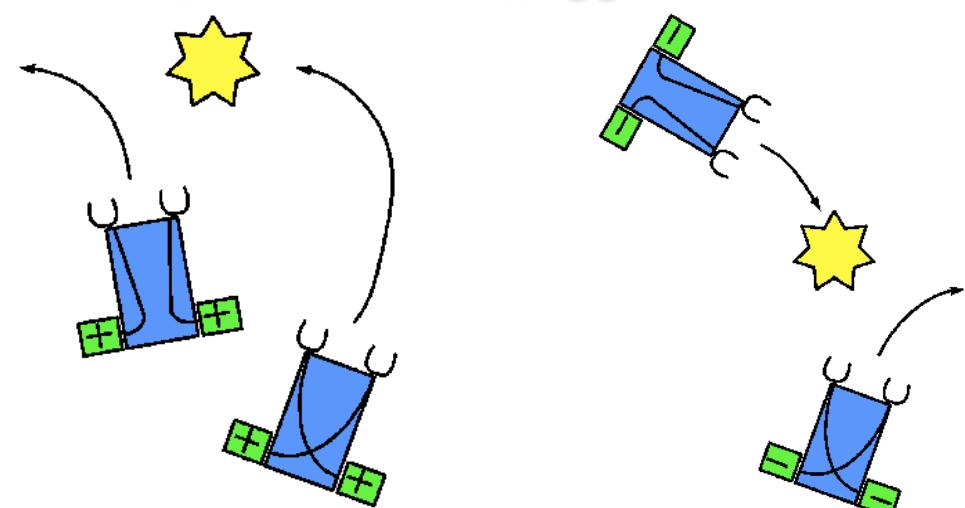
Archetypen der Sensor-Aktor-Kopplung → **Reflex**

- Ausgangspunkt: Differentialsteuerung auf Basis von Intensitätssensoren („Licht finden, Licht vermeiden“)
- Konstruktions-Freiheitsgrade:

- Verschaltung
- Beeinflussungsrichtung



„Aversion“ vs. „Aggression“



Bilder: <http://www.igi.tugraz.at/STIB/WS98/gruppe3/welcome.html>

# Hindernisvermeidung ohne Fahrtziel

**Reflex: Dreh Dich weg vom Hindernis!**



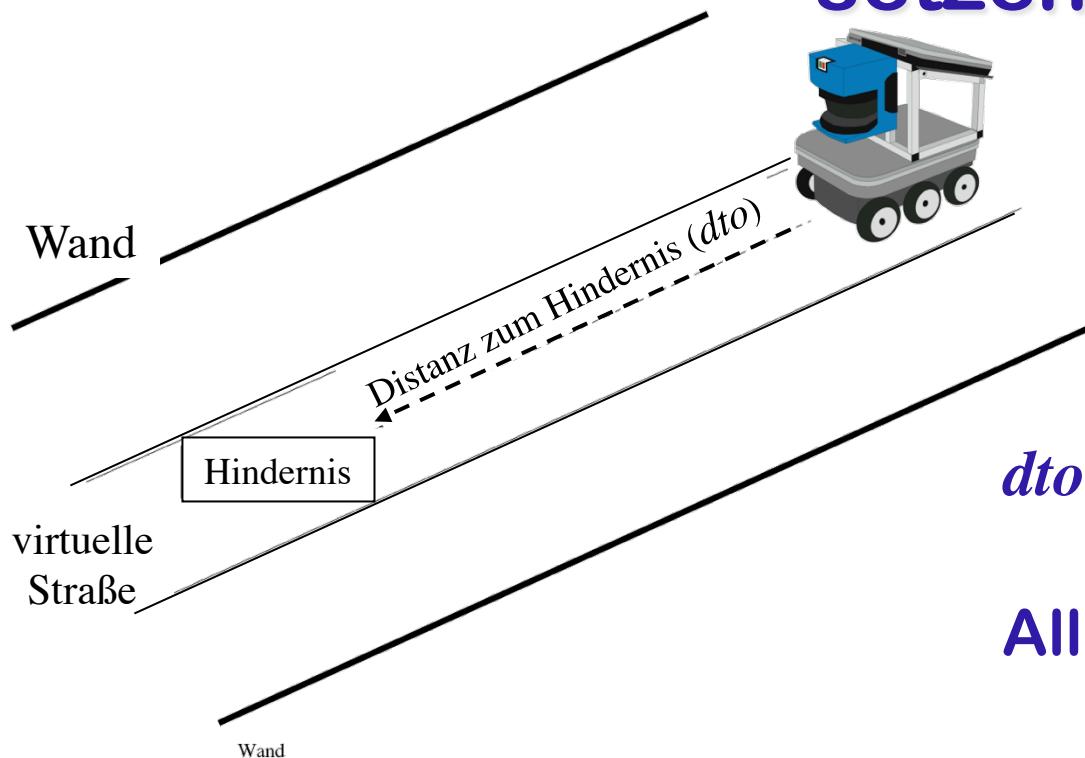
## Wohin? Wohin? Wohin?: Reaktion

- mit Kontaktsensorik (z.B. Roomba): Rücksetzen; festen Drehwinkel abhängig von Kontaktsektor
- mit Entfernungssensorik: Dreh/richte Dich in den Freiraum

## 7.3 Reaktive Navigation

- ... ist die Basis jedes Steuerungsprogramms für mobile Roboter
- ... wird seit Jahrzehnten immer wieder neu erfunden  
(Studierende, Schulkinder, ...)
- ... wird praktisch nicht (mehr) publiziert

# Hindernisumfahren I: Geschwindigkeit setzen



## Parameter

$$dto_{max} = 600\text{cm}, \quad dto_{min} = 70\text{cm}$$

$$v_{max} = 4\text{m/s}$$

$$dto < dto_{min} : v_{set} = 0$$

Alles frei bis  $dto_{max}$ :

$$v_{set} = v_{max}$$

$$\text{Sonst: } v_{set} = v_{max} \cdot dto/dto_{max}$$

**Achtung bei 2D Laserscanner:**  
virtual roadway nur in Scanebene einsehbar!

# Hindernisumfahren I: Beispiel

KURT2 drives with 3.5 m/s



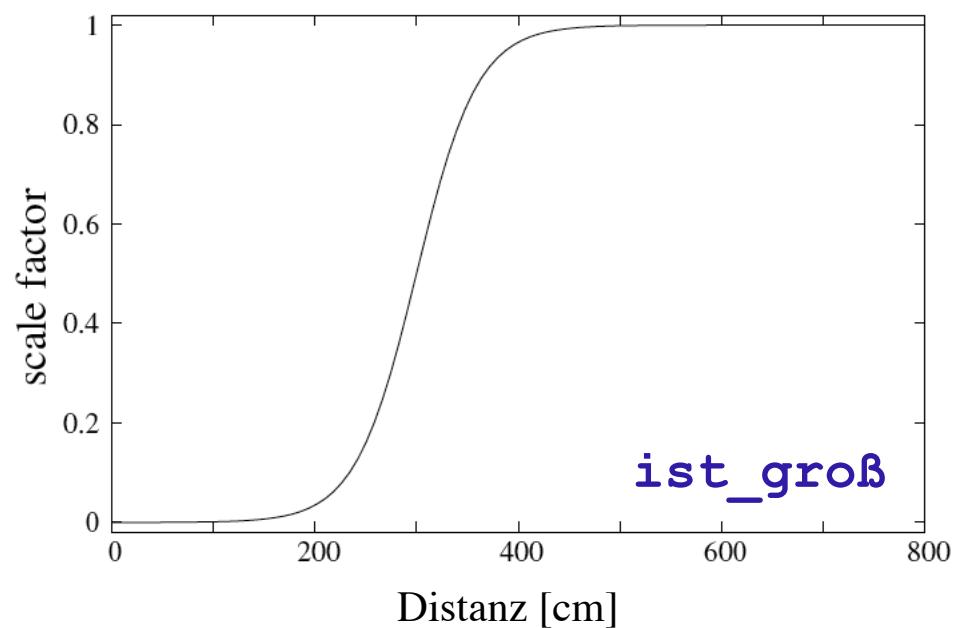
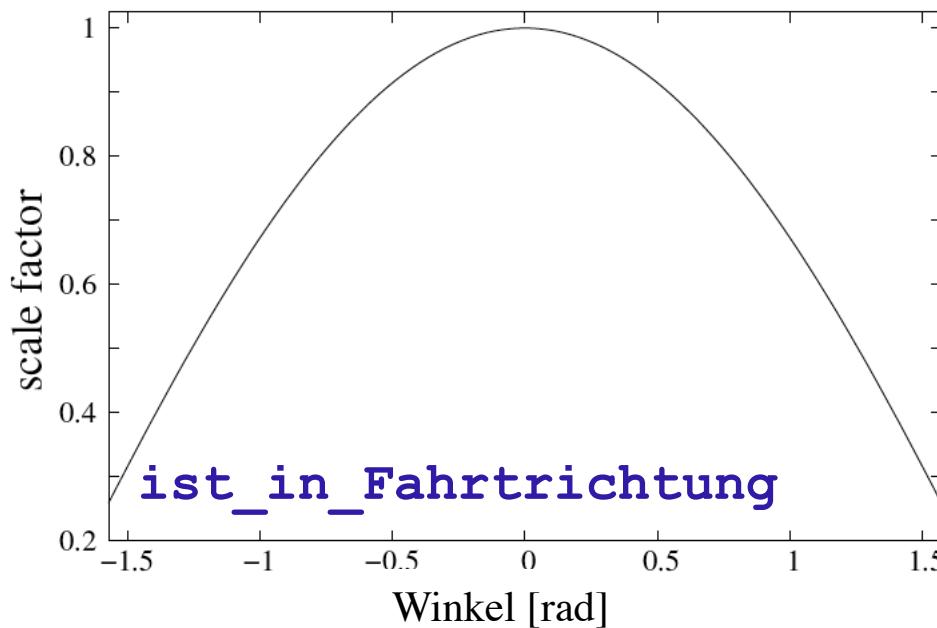
Für Schnellfahrt geeignet, sofern Laserscanner alle wesentlichen Hindernisse detektieren kann!

# Hindernisumfahren II: Richtung setzen (1/2)

**Basis:** |Scanpunktel| ( $i=0 \dots 180$ ) Fuzzy-Regeln der Form

WENN (Winkel<sub>i</sub> ist\_in\_Fahrtrichtung)  
UND (Entfernung<sub>i</sub> ist\_groß)  
DANN fahre in diese Richtung

UND pro Regel realisiert als Multiplikation d. Fuzzy-Werte



# Hindernisumfahren II: Richtung setzen (2/2)

## Zusammenrechnen der 181 Richtungs-, „Voten“

**Fusion** braucht man nicht nur bei Fuzzy-Regeln, sondern auch bei *Behaviors* und überall, wo Steuerungs-Parameter aus lokalen Beiträgen zusammengerechnet werden!

$$\theta_{set} = \text{atan2} \left( \begin{array}{l} \sum_{i=0}^{180} \sin(\varphi_i) \cdot w_{ori}(\varphi_i) \cdot w_{dist}(\varphi_i) \\ \sum_{i=0}^{180} \cos(\varphi_i) \cdot w_{ori}(\varphi_i) \cdot w_{dist}(\varphi_i) \end{array} \right)$$

Vorsicht, Falle!  
Scanwerte laufen von 0...180; Winkel laufen von -90...+90!

$w_{ori}$ ,  $w_{dist}$   
wichten Orientierung, Distanz entspr. Fuzzy-Prädikaten vorige Folie

**Weitere Falle:** Symmetrische Situationen! (Lösung z.B.: Rausch-Term)

# Konturverfolgung

## Beispiel für rechte Seite, Prinzip (Reflex)

- Eingabe: ideale Distanz  $d$ , Toleranz  $\varepsilon$
- bei der Vorwärtsfahrt in jedem Kontrollzyklus:
  - wenn gemessener Abstand rechts  $< d - \varepsilon$   
dann steuere „links“ im nächsten Zyklus
  - wenn gemessener Abstand rechts  $> d + \varepsilon$   
dann steuere „rechts“ im nächsten Zyklus

## Verbesserungen (Reaktion)

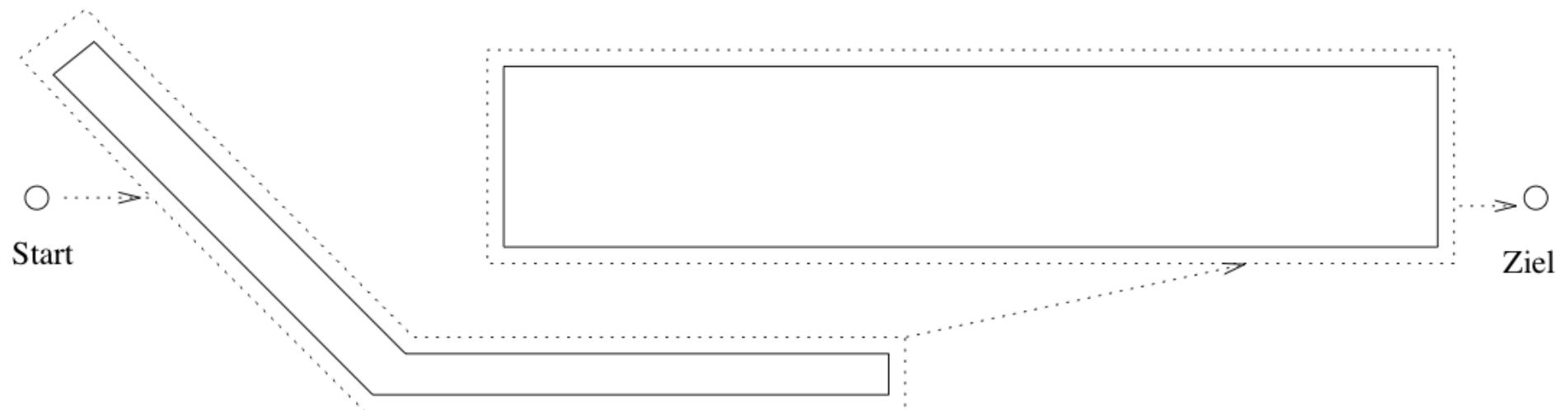
- Mittle über die letzten  $N$  Distanzmessungen (glätte Messfehler, Sprünge in den Daten  $\rightarrow$  **Vermeide Schwingungen in der Steuerung!**)
- Bei gerader Kontur (Wand) nutze zwei Abstandssensoren
- Winkeländerung hängt ab von der  $d$ -Abweichung  
(große Abweichung  $\rightarrow$  starke Lenkreaktion)

# Hindernisbewältigung bei Fahrtziel

**Bug-Algorithmen** (Lumelsky, 1990)

... sind nicht sehr clever, aber plausibel und erfolgreich!

## Bug 1



# Der Algorithmus Bug 1

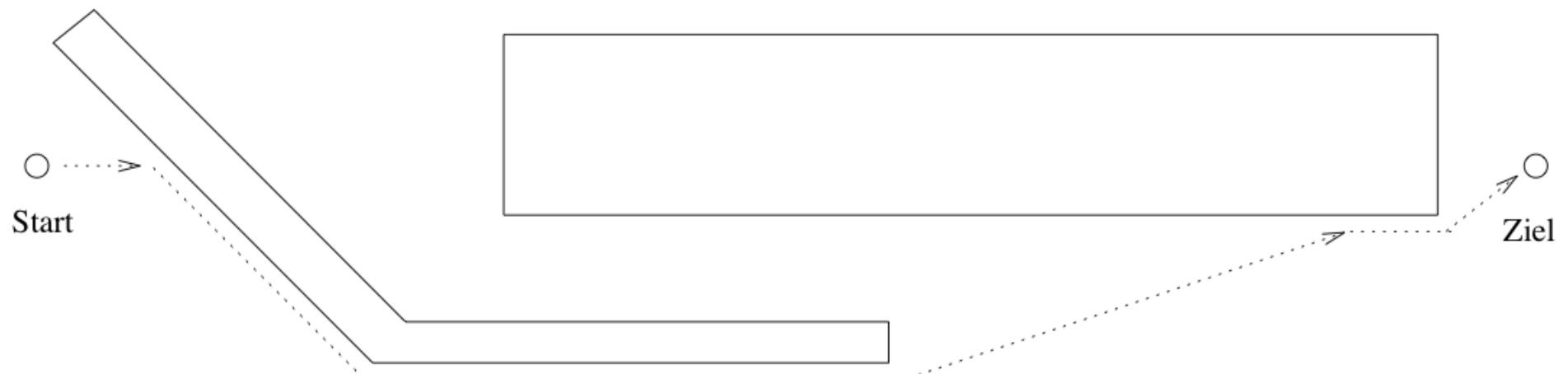
**Eingabe :** Richtung zum Ziel und Sensorwerte, die es erlauben, Hinderniskontakt festzustellen.

**Ausgabe:** Steuerbefehle, die vom Startpunkt zum Ziel führen.

```
1: repeat
2:   repeat
3:     fahre durch den Freiraum aufs Ziel zu
4:     terminiere, falls Ziel erreicht
5:   until Kontakt mit Hindernis
6:   k = aktuelle Position           // Kontakt mit Hindernis
7:   nz = k    // der bekannte Punkt auf Hinderniskontur, der dem Ziel am nächsten
      liegt
8:   repeat
9:     fahre entlang der Hinderniskontur
10:    if aktuelle Position liegt näher am Ziel als nz then
11:      nz = aktuelle Position
12:    end if
13:   until aktuelle Position == k
14:   fahre auf kürzestem Weg entlang der Hinderniskontur zu nz
15: until Ziel erreicht
```

# Bug mit Optimierung

- Bug 1 muss jedes im Weg liegende Hindernis komplett umrunden (Erwartungswert: 1 1/4-mal)
- Findet garantiert zum Ziel für endlichen Hindernisse
- Bug 3 umfährt das Hindernis nur, bis Weg zum Ziel frei
- Muss dafür erkennen, ob der frühere Weg gekreuzt wird



# Der Algorithmus Bug 3

---

**Eingabe :** Richtung zum Ziel und Sensorwerte, die es erlauben Hinderniskontakt festzustellen.

**Ausgabe:** Trajektorie, die vom Startpunkt zum Ziel führt.

- 1: **repeat**
  - 2:   **repeat**
  - 3:     fahre durch den Freiraum aufs Ziel zu
  - 4:     terminiere, falls Ziel erreicht
  - 5:   **until** Kontakt mit Hindernis
  - 6:   **repeat**
  - 7:     fahre entlang der Hinderniskontur startend in einer festen Richtung (z.B. rechts)
  - 8:     **until** Richtung zum Ziel ist frei und schneidet nicht den bisherigen Weg
  - 9: **until** Ziel erreicht
-

# Kompetitivität von online-Algorithmen

**online-Problem:** algorithmisches Problem, bei dem (einige) Eingabedaten erst zur Laufzeit bekannt werden. Z.B.

- Maschinenbelegungsplanung mit neuen Aufträgen
- Navigation in vorher (teilweise) unbekannter Umgebung

Wie misst man die Lösungsqualität eines online-Problems?

- Komplexität in Abh. von Eingabe klappt nicht
- **Kompetitivität:** Vergleich der garantiert erzielbaren online-Lösung mit der (theoretischen) optimalen Lösung

online-Verfahren  $V$  mit Kosten  $K_V$  **kompetitiv mit**  
endlichem **Faktor  $C$ :**

$$K_V \leq C \cdot K_{opt} + C'$$

# Sind Bug-Strategien kompetitiv?

- Alle Bug-Algorithmen umrunden alle Hindernisse, auf die sie treffen (ganz oder teilweise)
- Ist die Maximalzhl  $M$  der Hindernisse und ihr maximaler Umfang  $U$  endlich?
- Dann ist der Umweg von Bug 1 begrenzt durch ca.  $M \cdot \frac{3}{2} U$
- Das ist ein arg schlechter Wert
- Die praktische Alternative in der Robotik ist nicht eine kompetitivere online-Strategie bei unbekannter Umgebung, sondern Navigation mit Karte (Pfadplanung)
- Reaktive Navigation (mit Bug oder anders) bleibt eine wichtige Zugabe für nicht kartierte Hindernisse!