
Programmierung von eingebetteten Systemen **— LEGO-Mindstorms —**

Reinhard von Hanxleden Stephan Höhrmann Jan Lukoschus
Alwin Stengel Martin Steffen

*Inst. für Informatik u. Prakt. Mathematik
Christian-Albrechts Universität zu Kiel*

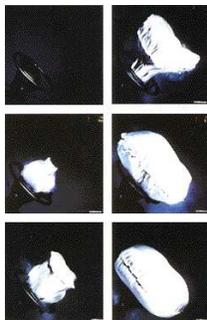
Was sind eingebettete Systeme?

- Kombination von **Hard- und Software**, meist für *eine* bestimmte Aufgabe, als
- **Teil** eines größeren Systems, welches kein “Computer” sein muß.

Wo finden sich eingebettete Systeme?

Überall

- Unterhaltungselektronik (DVD, CD, ...)
- Handys, Kameras, Haushaltsgeräte ...
- Autos (Airbag, Motorsteuerung, ...)
- Industrielle Prozesse, Fertigungsautomatisierung
- Luft- und Raumfahrt



...

Eingebettete Systeme sind **sehr verschieden**

- oft **sicherheitskritisch**
- **Echtzeitsysteme**
- Interaktion mit der physikalischer Umwelt, (“**Reaktive Systeme**”)
- oft bestehend aus verschiedenen **kommunizierenden/interagierenden** Komponenten (“**verteiltes**” System)

Was bedeutet Echtzeit und Echtzeitsysteme?

- Ein **Echtzeitsystem** ist eines dessen *Korrektheit* von
 - den errechneten **Ergebnissen** abhängt als auch vom
 - **Zeitpunkt**, wann diese geliefert werden
- ein **zu spätes** (oder **zu frühes!**) Ergebnis: genauso “falsch” wie ein fehlerhaft Berechnetes ... oder schlimmer
- Echtzeit \neq “sehr schnell”
- eingebettete Systeme: fast immer **Echtzeit**

Programmierung eingebetteter Systeme

- traditionelle **Progammiersprachen** nicht unbedingt geeignet
- traditionelle **Betriebssysteme** nicht unbedingt geeignet
- Interaktion mit der Umwelt: **komplex**
- Interaktion von Teilsystemen untereinander: **komplex**
- Software reagiert nicht **gutartig** was Fehler betrifft („**kleine Ursache, grosse Wirkung**“)

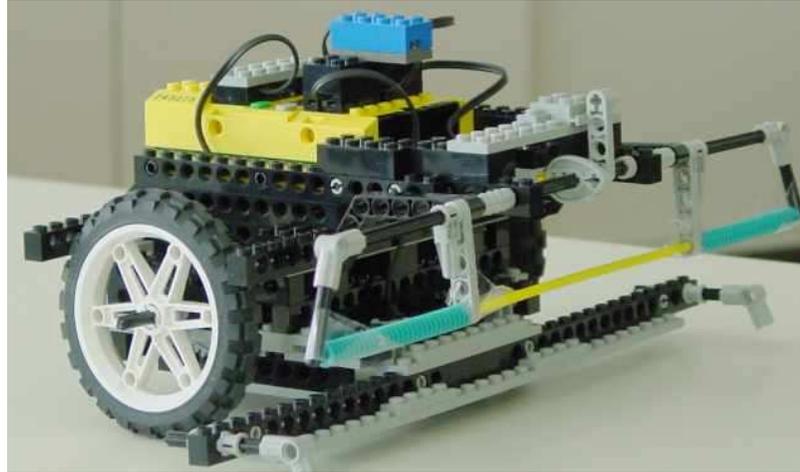
Programmierung eingebetteter Systeme

- Software reagiert nicht **gutartig** was Fehler betrifft („**kleine Ursache, grosse Wirkung**“)

4. Juni 1996, Kourou, Franz. Guayana, direkte Kosten: ca. 1 G-Euro



Was hat LEGO-Mindstorms mit all dem zu tun?



- “leicht zugängliches”, relativ erschwingliches eingebettetes System
- frei programmierbar
- verschiedene Programmiersprachen verfügbar
- viele Programmier-/Design-Probleme “in nuce” studierbar

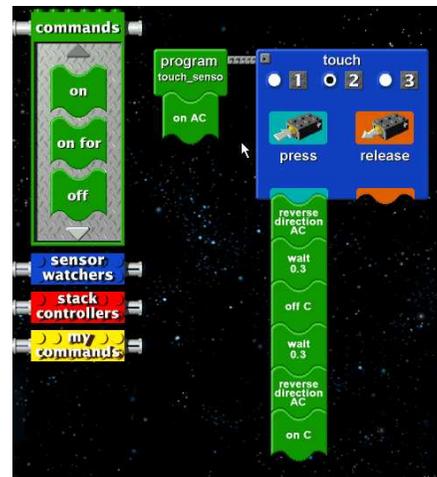
LEGO-Mindstorms: Hardware



- „RCX“
- LEGO-Bausteine (Zahnräder, Stangen, Achsen, ...)
- Schnittstelle zur Aussenwelt:
 - **Aktoren**: 2 Motoren,
 - **Sensoren**: 2 Drucksensoren, Lichtsensor
 - **Infrarotschnittstelle** (Kommunikation, Download)

Software: RCX-Code

- **visuelle** Programmierumgebung
- bei LEGO-Mindstorms mitgeliefert
- für **Laien** geeignet
- Siehe auch den Vortrag „*Programmierung von Kleinrobotern*“ von Hanus



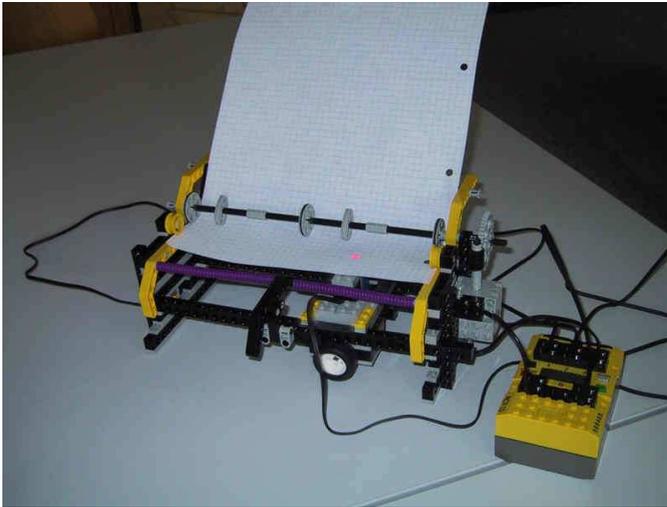
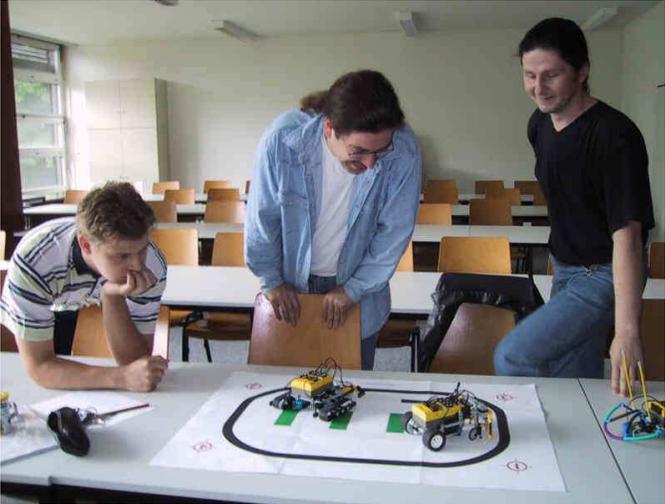
- LEGOS
 - freies (und kostenloses) Betriebssystem
 - arbeitet mit Linux/Unix/Windows zusammen
 - verschiedene Programmiersprachen:
 - C
 - Java
 - Curry
 - Esterel/Estudio
 - ...

Siehe Vortrag Hanus et. al.

Due: Thu, 13 Dec 2001, 15:00 hrs, per E-mail to sho@informatik.uni-kiel.de , CC: rvh@ ... Late submission: possible, but -5 points

1. Build a **robot** that **moves** forward, and expects a heartbeat **signal** from the base station within every **100 msec**. The heartbeat should be encoded as a message containing a single byte; default addresses for the host and the RCX; port 9. It should **tolerate** missing heartbeats for up to 1 sec, but sound an **alarm** if heartbeats are missed. After that, ...
 - (a) Documentation (overview of approach and assumptions, comments) (3 pts)
 - (b) Functional robot (2 pts)
2. For the robot, **analyse** its timing behavior ...

Einsatz in der Lehre



Programmier-Konzept: Wie beschreibe ich das Verhalten eines System?

- Es sich in einem von mehreren **Zuständen** befinden
- Es kann seinen Zustand **wechseln**:
 - als **Reaktion** auf **Umweltereignisse**
 - nach bestimmter **Zeit**
- Es kann die Umgebung **beeinflussen**

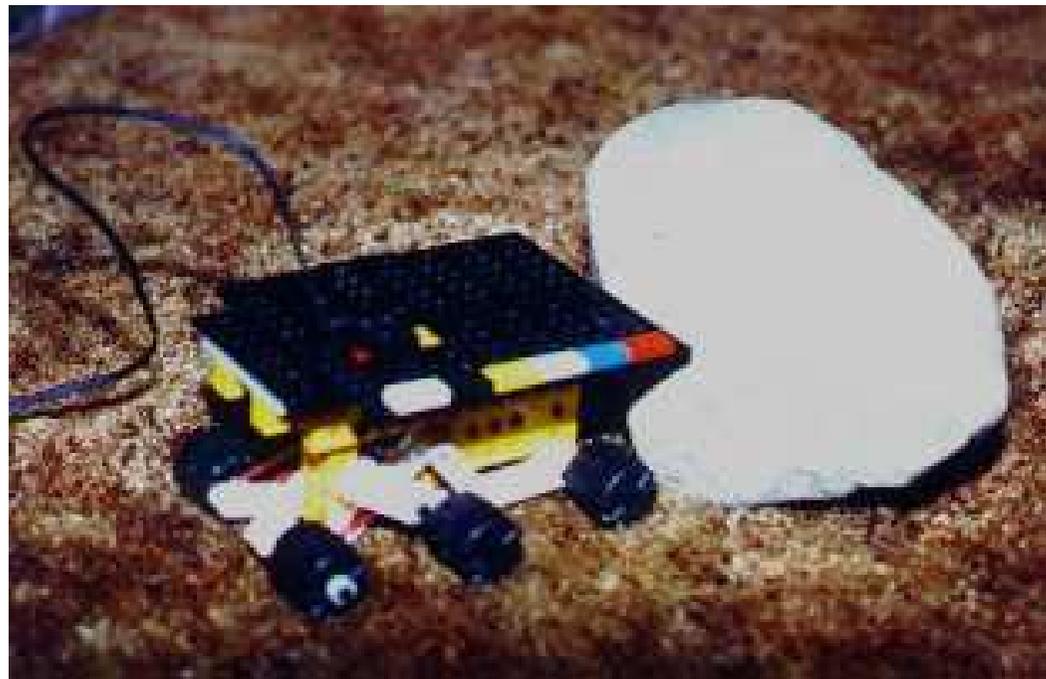
⇒ **Zustands/Übergangsmodell**

- in graphischer Darstellung: “Kreise” und “Pfeile”

Beispiel: “Bumper”

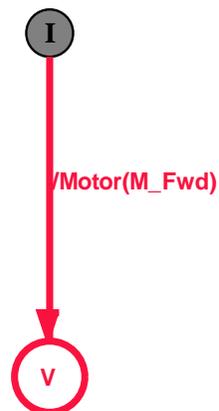
Ziel: autonomer Roboter der Hindernissen ausweicht

- Sensoren: Berührungssensor für rechts + links
- Aktoren: Motorantrieb (Vorwärts, Rückwärts, Aus, Stopp, Links, Rechts)

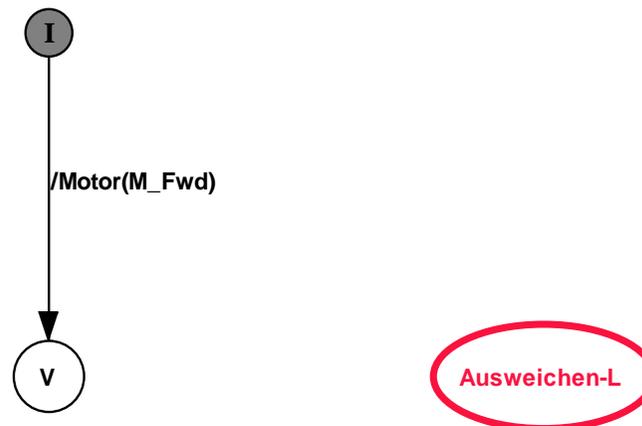


1. fahre **vorwärts**
2. wenn Du gegen **Hindernis stößt**, dann **weiche aus** und zwar
 - (a) bei Kontakt **rechts**, nach **links**
 - (b) bei Kontakt **links**, nach **rechts**
3. dann: fahre wieder **vorwärts**

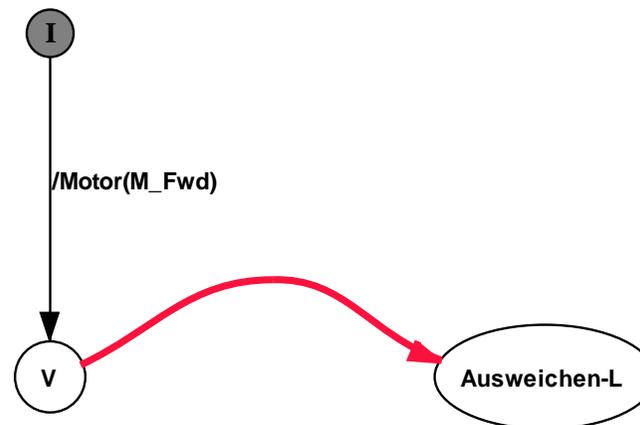
Beginne durch: **Motoren einschalten** (vorwärts)



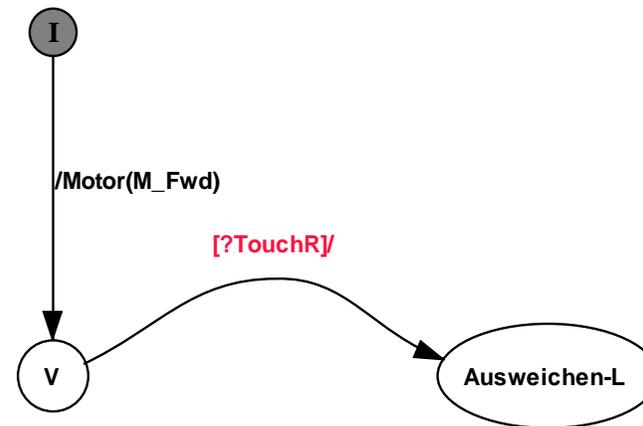
Mögliches Verhalten: **Ausweichen** nach links



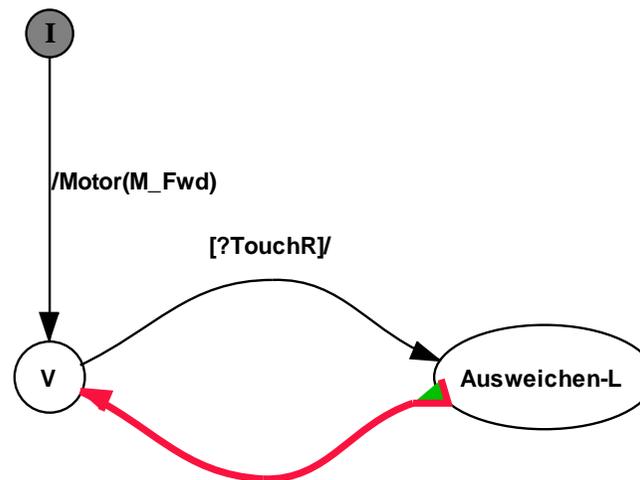
Umschalten von Vorwärts nach Ausweichen



Wann nach links ausweichen?: Wenn der **rechte** Schalter gedrückt wird



wieder **vorwärtsfahren**, wenn das Ausweichen **fertig** ist

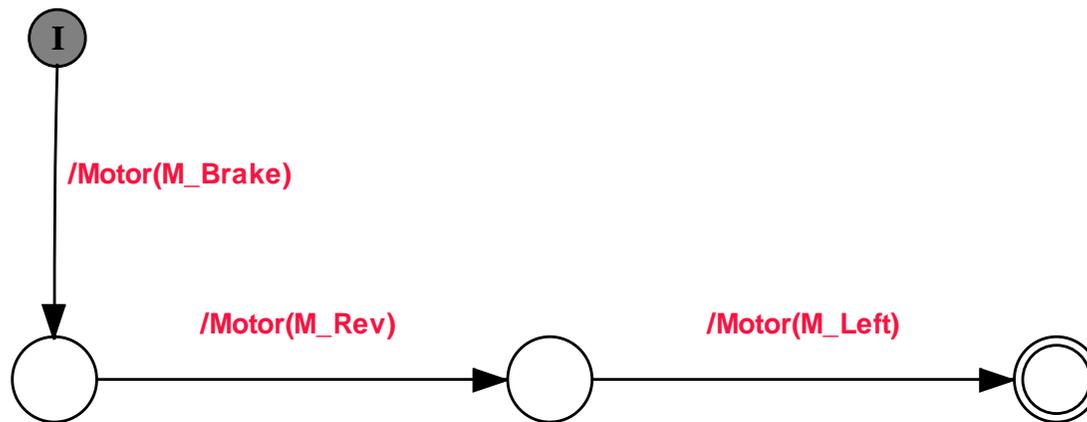


Nur: Wie geht Ausweichen-Links?

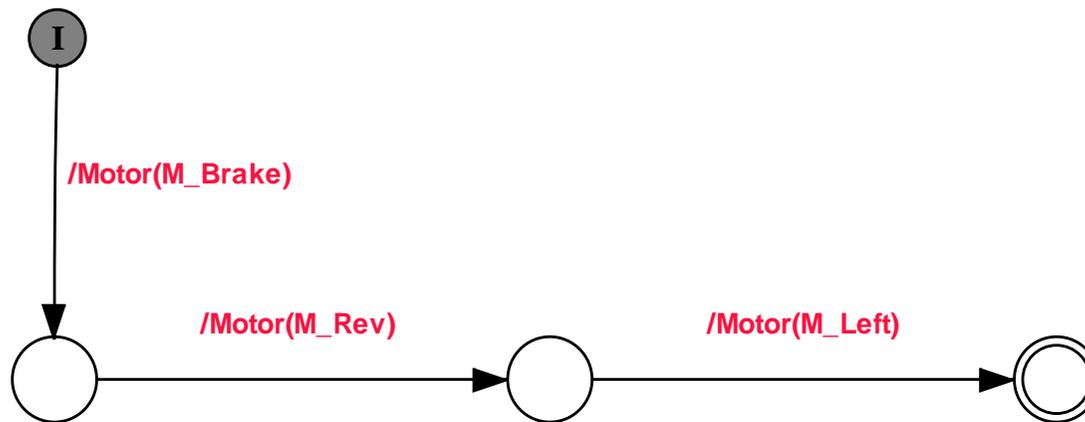
Tue der Reihe nach die Schritte:

1. **stehenbleiben** (M_Brake)
2. **zurücksetzen** (M_Rev)
3. nach **links drehen** (M_Left) und

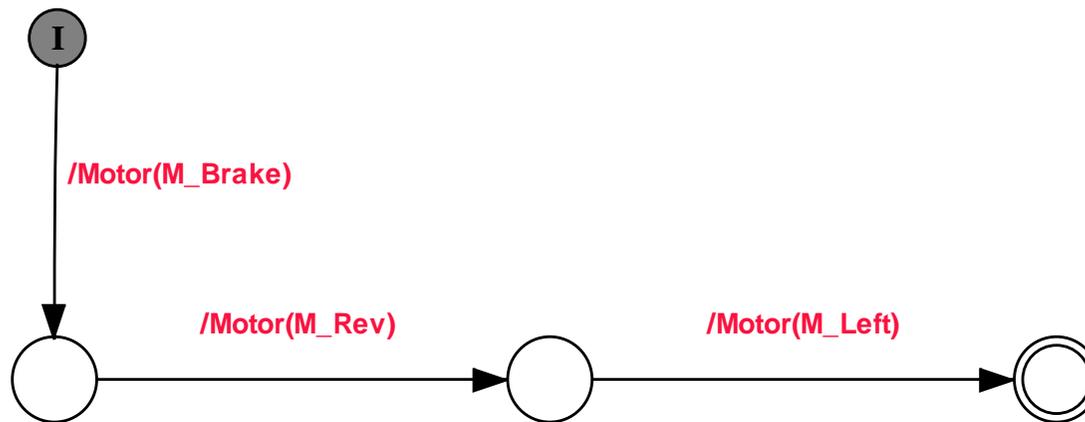
Ausweichen



Ausweichen: ist das *Alles*?



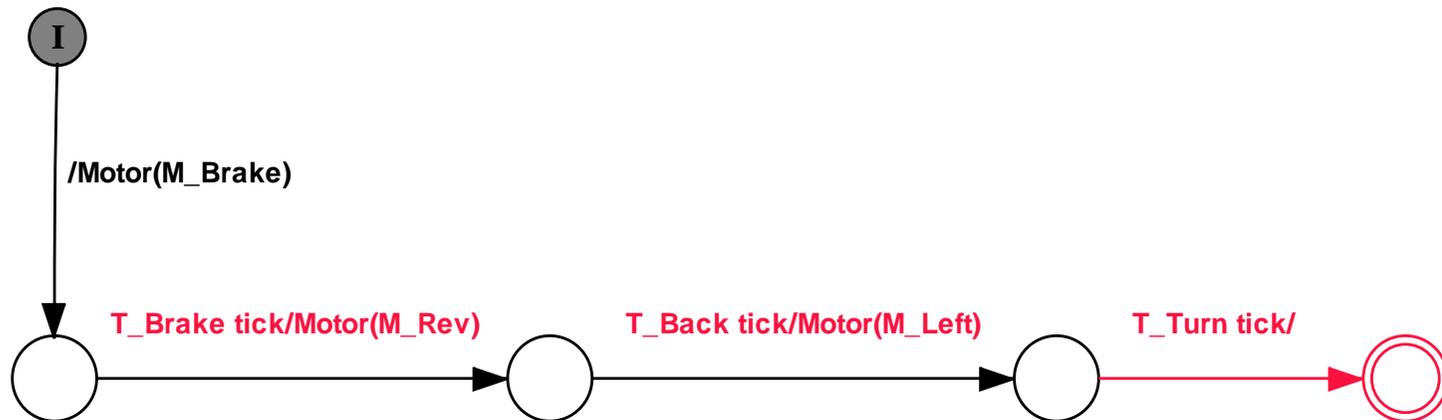
Ausweichen: ist das *Alles*?



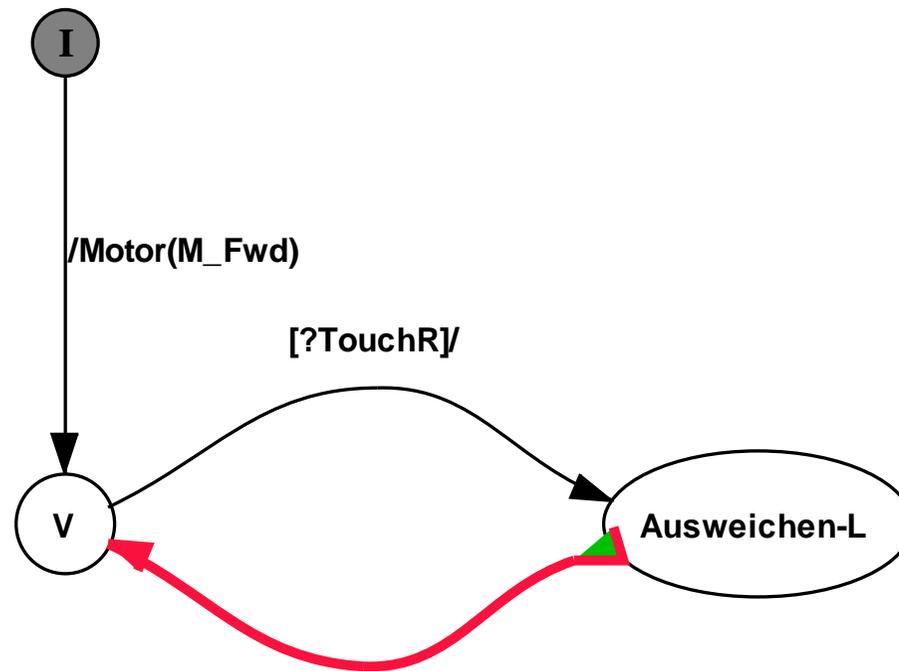
die **Zeit** fehlt!

Warte 5 Ticks/Taktzyklen bevor Du in den nächsten Zustand gehst (und dabei die Motoren nach *links-drehen* schaltest) \Rightarrow

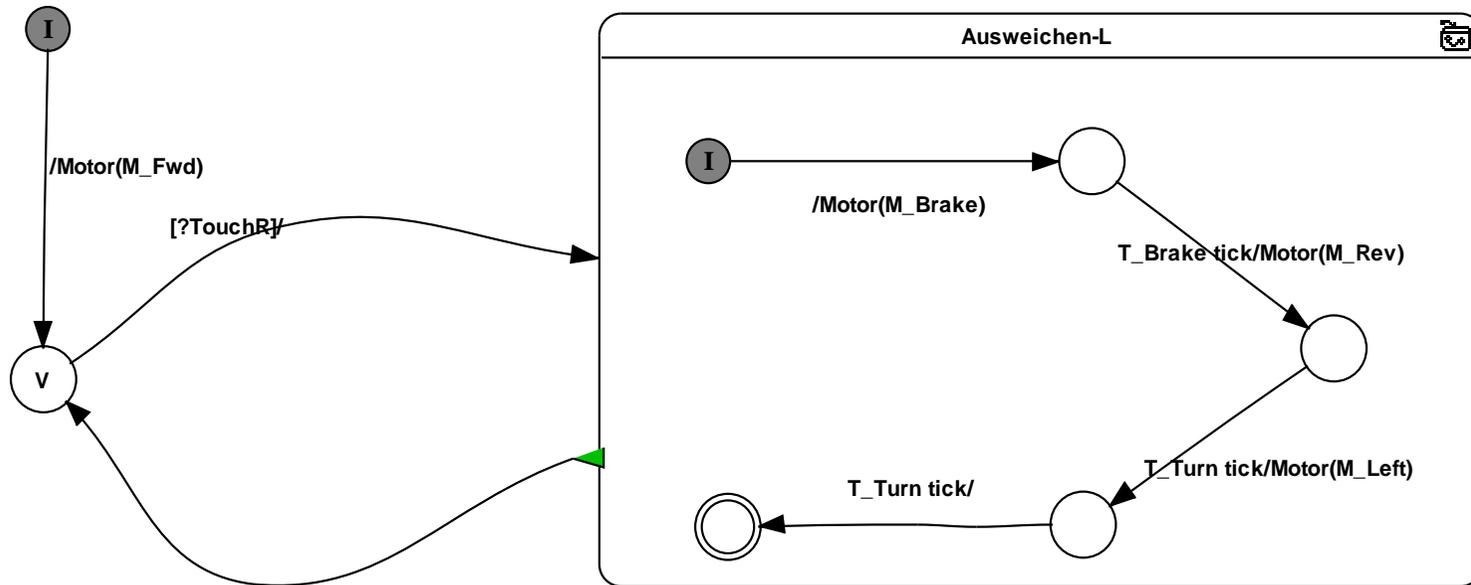
5 tick/Motor(M_Left)



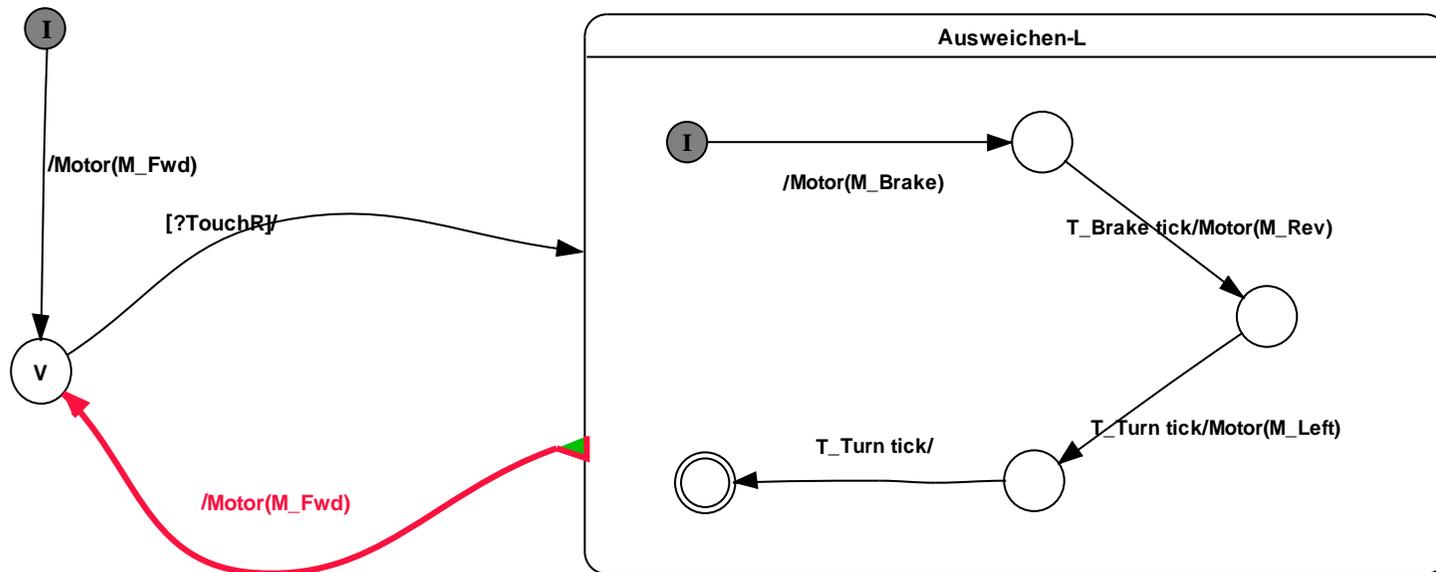
Zusammen mit "Vorwärts"



Zusammen mit “Vorwärts”: Zoom in

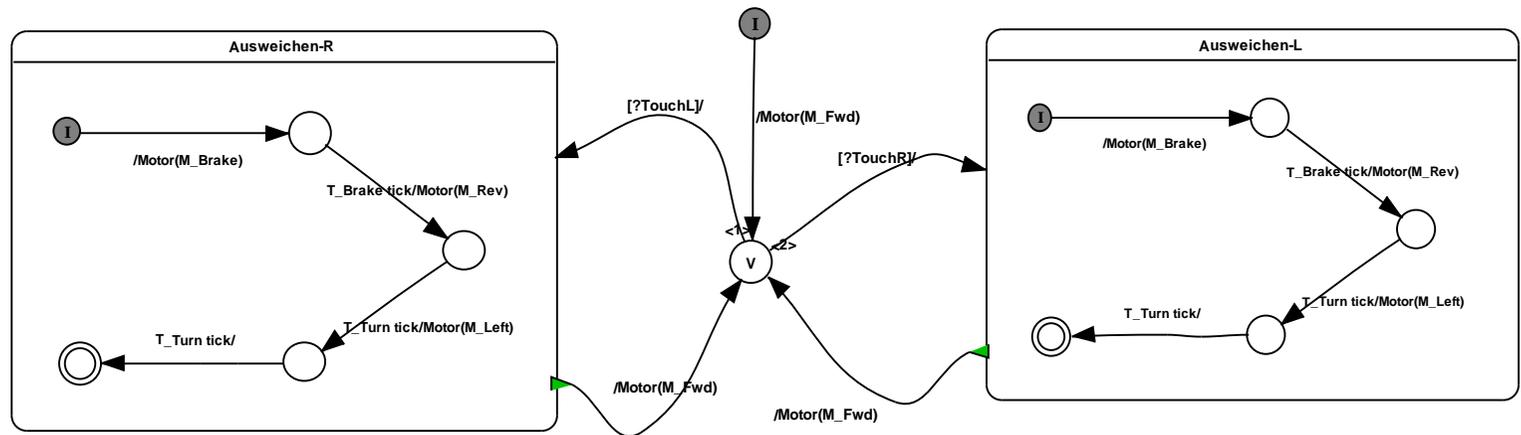


Zusammen mit “Vorwärts”: Zoom in



Motoren wieder auf **vorwärts** stellen!

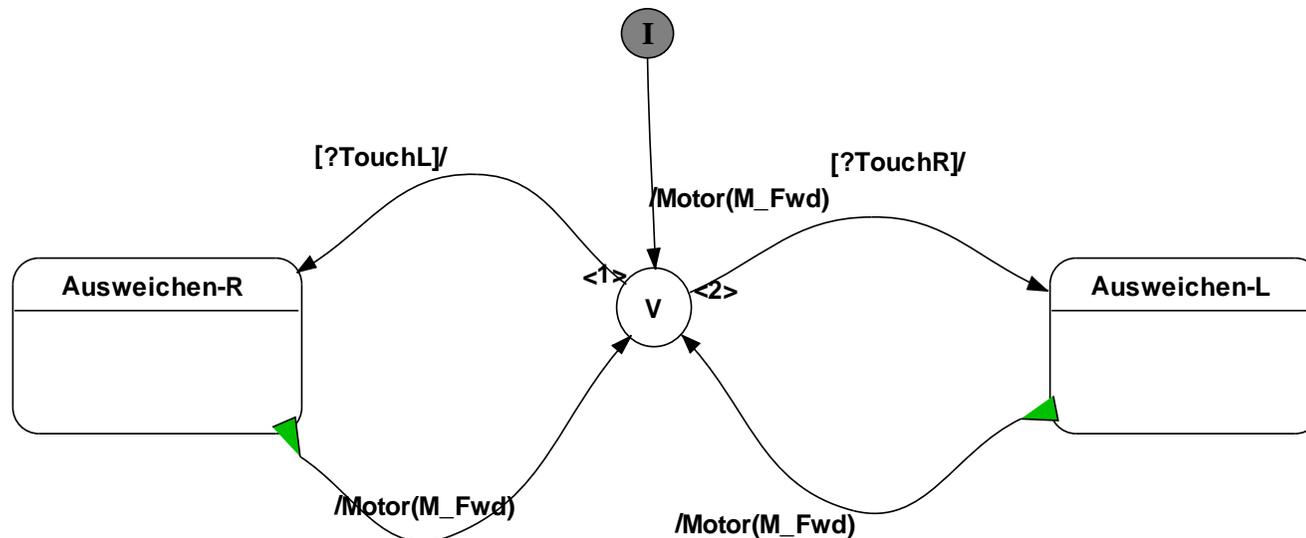
Die 2te Hälfte: entsprechend



Bumper: Verbergen von Details

Auf oberster Ebene: 3 Zustände

- Vorwärts
- links Ausweichen
- rechts Ausweichen



Es gibt noch viel mehr

- Wie kriegt man die wachsende **Komplexität** der Systeme in den Griff?
- Wie sehen geeignete **Programmiersprachen** aus?
- Wie entwickelt man **Hard- und Software** zusammen?
- Wie **modelliert** man digitale Systeme in physikalischer (= nichtdigitaler) Umwelt?
- Wie garantiert man **Zuverlässigkeit/Sicherheit** eingebetteter Systeme?
 - Testen
 - Verifikation

⋮

Literatur

- [1] Gerard Berry. *The Esterel v5 Language Primer (Version v5..91)*. Centre de Mathématiques Appliquées Ecole Mineur and INRIA, July 2000.
- [2] Esterel Technologies. *Esterel Studio V3.1, Reference Manual*, October 2001.
- [3] Christoph Kirsch. Embedded software engineering. Slides for a spring course, 2001, EECS Department UC Berkely, 2001. available at <http://www.eecs.berkeley.edu/fresco/giotto/course>.
- [4] Jonathan B. Knudsen. *The Unofficial Guide to LEGO Mindstorms Robots*. O'Reilly, first edition, October 1999.
- [5] Hermann Kopetz. *Design Principles for Distributed Embedded Application*. Kluwer Academic Publishers, 1997.
- [6] Phillipe Lacan, Jean Noël Monfort, Le Vinh Quy Ribal, Alain Deutsch, and Gearges Gonthier. Ariane 5. the software reliability verification process: The Ariane 5 example. In *Proceedings of DASIA'98 (Data Systems in Aerospace)*. ESA Publications, 1998.
- [7] Reinhard von Hanxleden. Real-time systems programming. Lecture, Summer Term 2002, 2002. available at www.informatik.uni-kiel.de/inf/von-Hanxleden/teachi