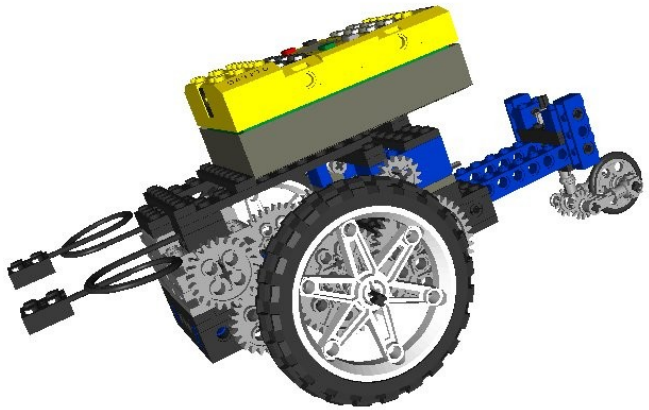
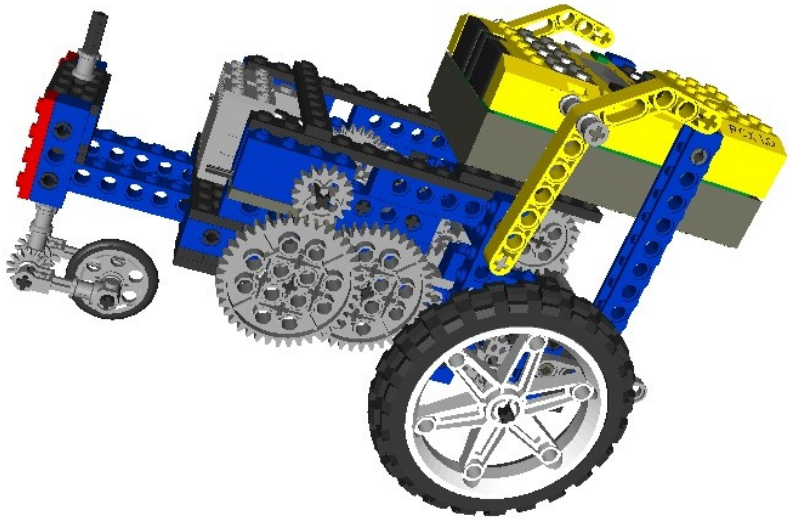
HOCHSCHULE harz

lejON





lejON – LeJOS Odometric Navigator

Seminarvortrag zum Teamprojekt
im Fach „Programmierung mobiler Agenten“
für den Masterstudiengang „Informatik/Mobile Systeme“
im Wintersemester 2005/2006 an der Hochschule Harz

Referenten

Jan Grohmann u20189@hs-harz.de
Annedore Rößling u20183@hs-harz.de
Florian Ruh u20184@hs-harz.de
Kai Schories u20187@hs-harz.de

Homepage

http://lejon.florianruh.de/

Agenda

- Motivation

  Aufgabenstellung, Ziele, Grenzen von RCX und LeJOS

- Architektur

  Odometrie, Single und Dual Differential Drive, DIDI und HILDE

- lejON-API

  Schichten, Anwendungsfälle, Klassen, Methoden und Zustände

- Evaluierung

  Testparcours und Testläufe

- Fazit

  Zusammenfassung und Ausblick

Motivation

Aufgabenstellung

- Entwicklung einer Architektur zur Navigation von zweirädrigen Robotern mit LeJOS
  - LeJOS = Lego Java Operating System
  - Java VM als Firmware-Ersatz auf dem LEGO-Mindstorms RCX-Baustein
- Ziele
  - Kapselung der Navigationsfunktionen
  - Anwender nutzt High-Level-Methoden statt LeJOS-API direkt
  - Präzisere Navigation als mit *RotationNavigator* aus LeJOS
  - SW-gestützte Regelung der Kursabweichung

Grenzen von Mindstorms

- RCX
  - Hitachi-8-bit-MC mit 16 MHz, 32 KB RAM
  - Nur 12 KB des RAM für User Code
  - Weniger geeignet für Echtzeitanwendungen
- Weitere Probleme
  - Nur 8 Leistungsstufen der Motoren
  - Grobe Auflösung der Rotationssensoren (16 Ticks/U)
  - Geringe Verarbeitungsbreite

Grenzen von LeJOS

- Klasse *RotationNavigator*
  - Unpräzise Navigation
  - Nur Translation und Rotation möglich
  - Keine Kurvenfahrt
  - Keine eigene API
- Effiziente Ausnutzung der Ressourcen nötig
  - Möglichst wenige Objekte, da kein GC vorhanden
  - „Schlanker" Code für max. 12 KB
- Minimum an Packages vorhanden
- Keine long-Arithmetik

Architektur

Odometrie 1

- Odometrie: Berechnung von Wegstrecke und Richtungswinkel
  - Ermittlung über Rotationssensoren
  - Impulszählung
  - Ermittlung von Umrechnungsfaktoren
    - Ticks pro Zentimeter (TPC)
    - Differentielle Ticks pro Grad (DTPD)
  - Berechnung der aktuellen Position und der Zielkoordinaten
  - → Kursabweichungen erkennbar

- Berechnung der aktuellen Position
  - Ermittlung der Skalierungsfaktoren

$$TICKS_{\text{effective}} = ratio_{\text{gear}} \cdot TICKS_{\text{nominal}}$$

$$TPC = \frac{TICKS_{\text{effective}}}{\pi \cdot \text{diameter}_{\text{wheel}}}$$

$$DTPD = \frac{\pi \cdot \text{length}_{\text{axle}}}{360} \cdot TPC$$

  - Ermittlung der zurückgelegten Bewegung

$$\Delta s = \frac{TICKS_{\text{effective}}}{TPC}$$

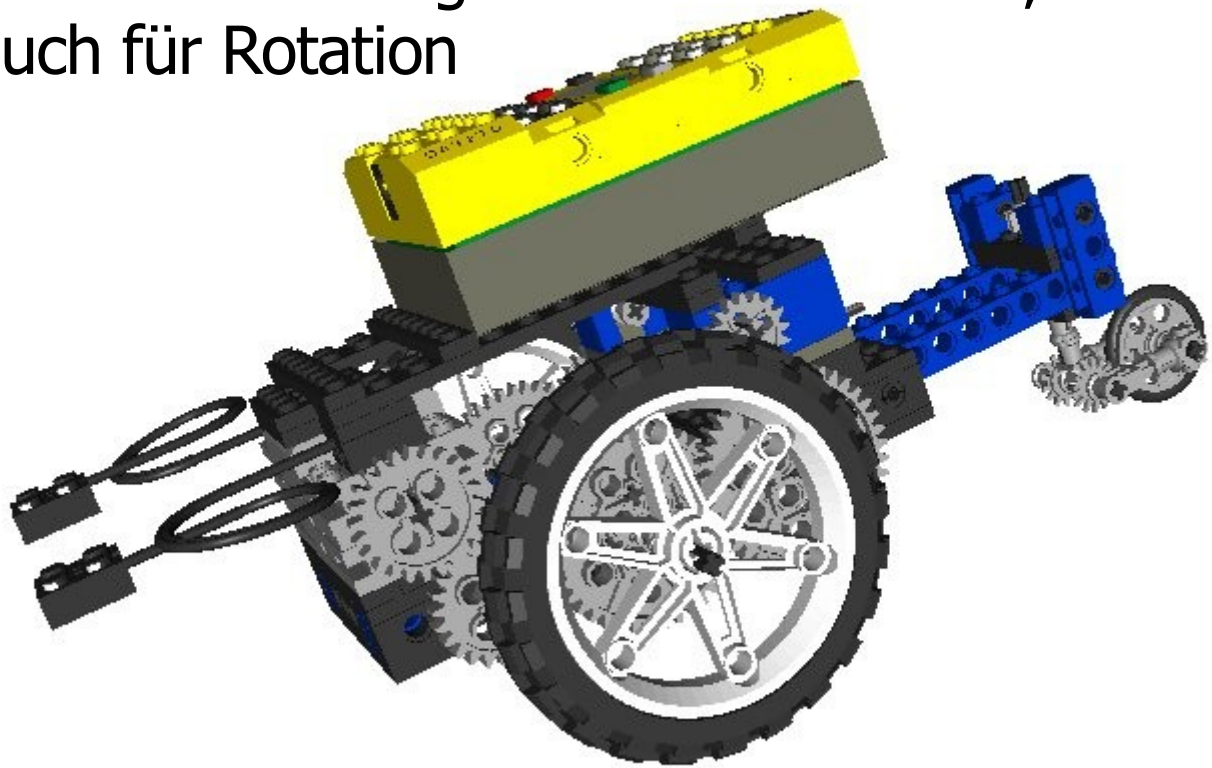
$$\Delta \vartheta = \frac{(\Delta TICKS_{\text{effective r}} - \Delta TICKS_{\text{effective l}})}{DTPD}$$

Hardware-Architektur

- Beliebige Hardware-Architekturen möglich
- Voraussetzungen und Einschränkungen
  - Kunststoffbauteile
  - Zweirädriger Antrieb
    - kleine Räder
    - wenig Reibung
  - Stützrad
    - möglichst unbelastet
    - schwerster Baustein (RCX) muss über Antriebsachse sein
- Hier behandelt
  - Single Differential Drive (DIDI)
  - Dual Differential Drive (HILDE)

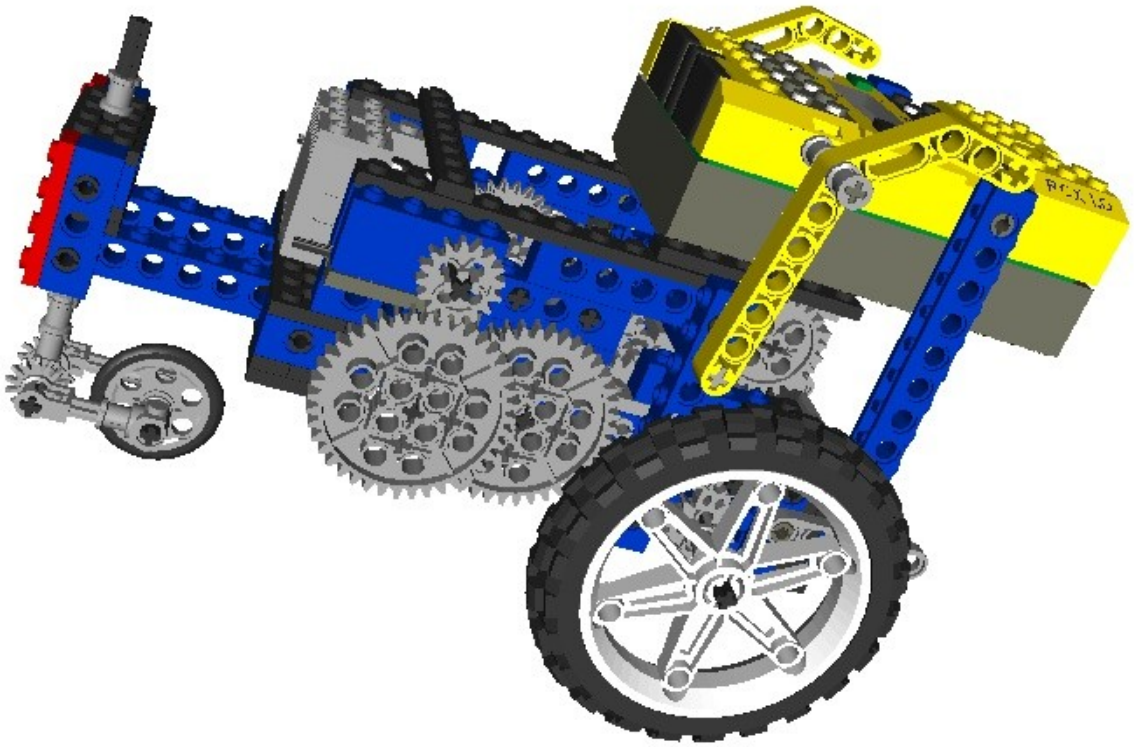
DIDI

- DIDI = Differential Drive
  - Prototyp für ein Single DD
  - Voneinander getrennte Antriebsräder
  - Ein Motor pro Rad
  - ➔ Zwei Motoren zuständig für Geradeausfahrt, beide auch für Rotation

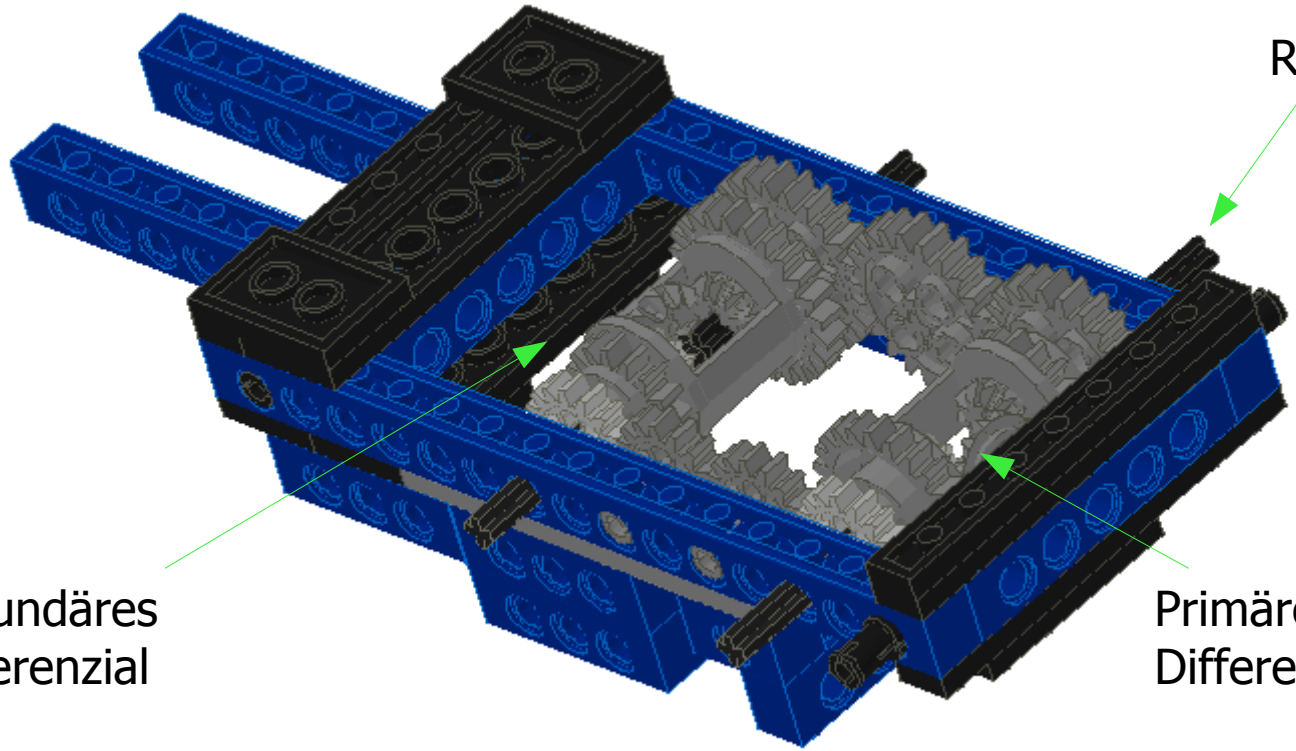


- Problem/Herausforderung
  - Gleichmäßige Bewegung gestört durch
    - Unterschiedliche Motorenbeschaffenheit (Herstellungsfehler, Verschleiß)
    - Spiel der Zahnräder
    - Kunststoff als Material aller Bauteile
  - Navigation nicht ohne Regelung möglich!

- HILDE = High-Intelligence Lego
  Dual Differential Drive Engine
  - Prototyp für ein Dual DD (nach /2/)
  - Antriebsräder mit zwei Differenzialen verbunden



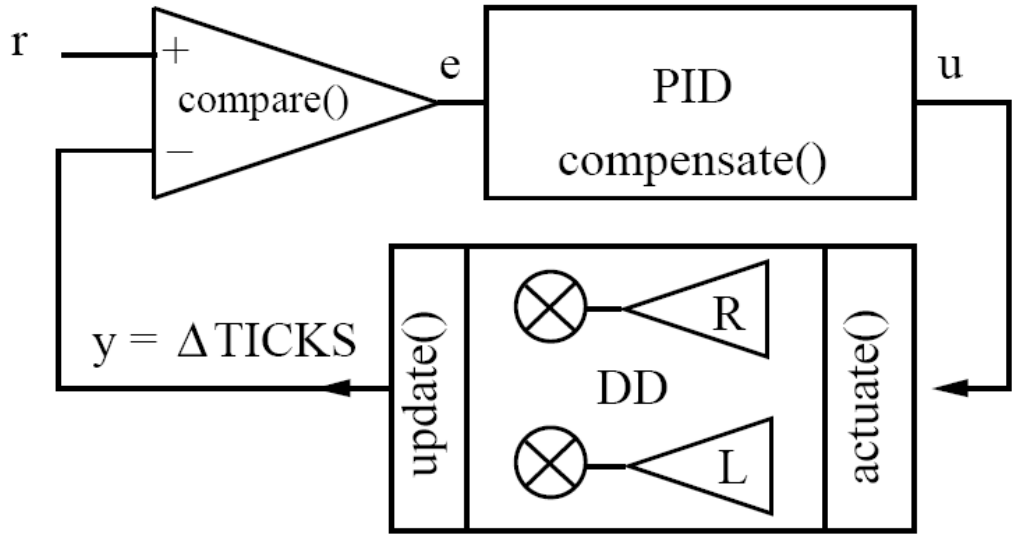
- Ein Motor fürs Primäre Differenzial
- Einer fürs Sekundäre Differenzial
- ➔ Nur ein Motor zuständig für Geradeausfahrt, einer für Rotation
- ➔ Möglichkeit der Bewegungsüberlagerung



- Vorteile
  - Unterschiedliche Motorenbeschaffenheit wirkt sich nicht aus
  - Idealfall: keine Regelung notwendig
- Problem/Herausforderung
  - Gleichmäßige Bewegung gestört durch
    - Spiel der Zahnräder
    - Kunststoff als Material aller Bauteile
  - Bewegung nicht ohne Regelung möglich

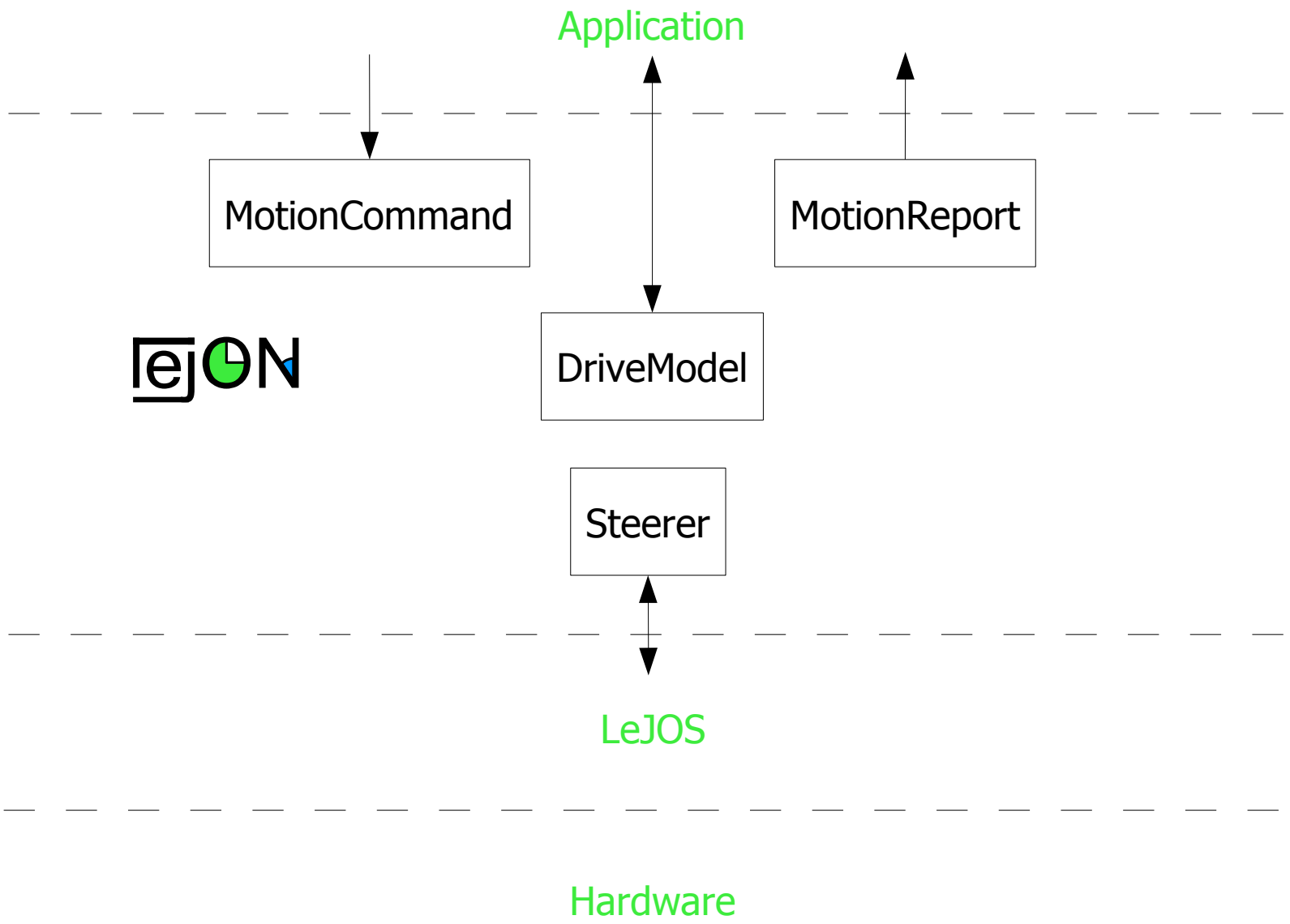
PID-Regler

- Regelung der Stellgröße
  - Vergleich Soll-/Ist-Werte
  - Beruht auf Anzahl der Ticks
  - PID-Konstanten einzeln für jede Architektur berechnen
  - Verfahren nach Ziegler-Nichols

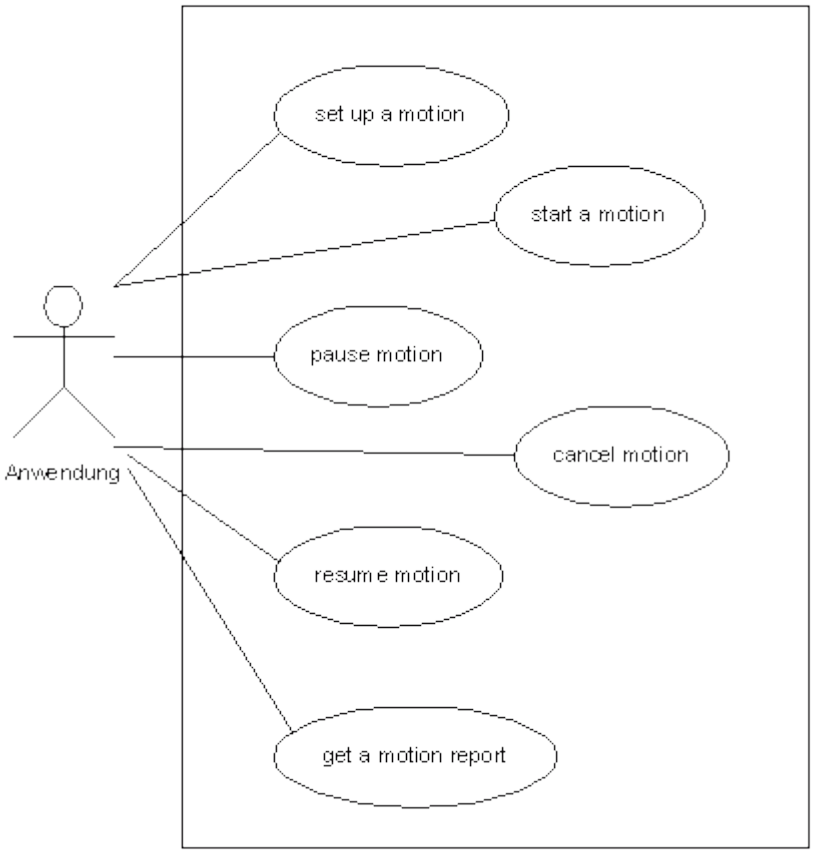


lejON-API

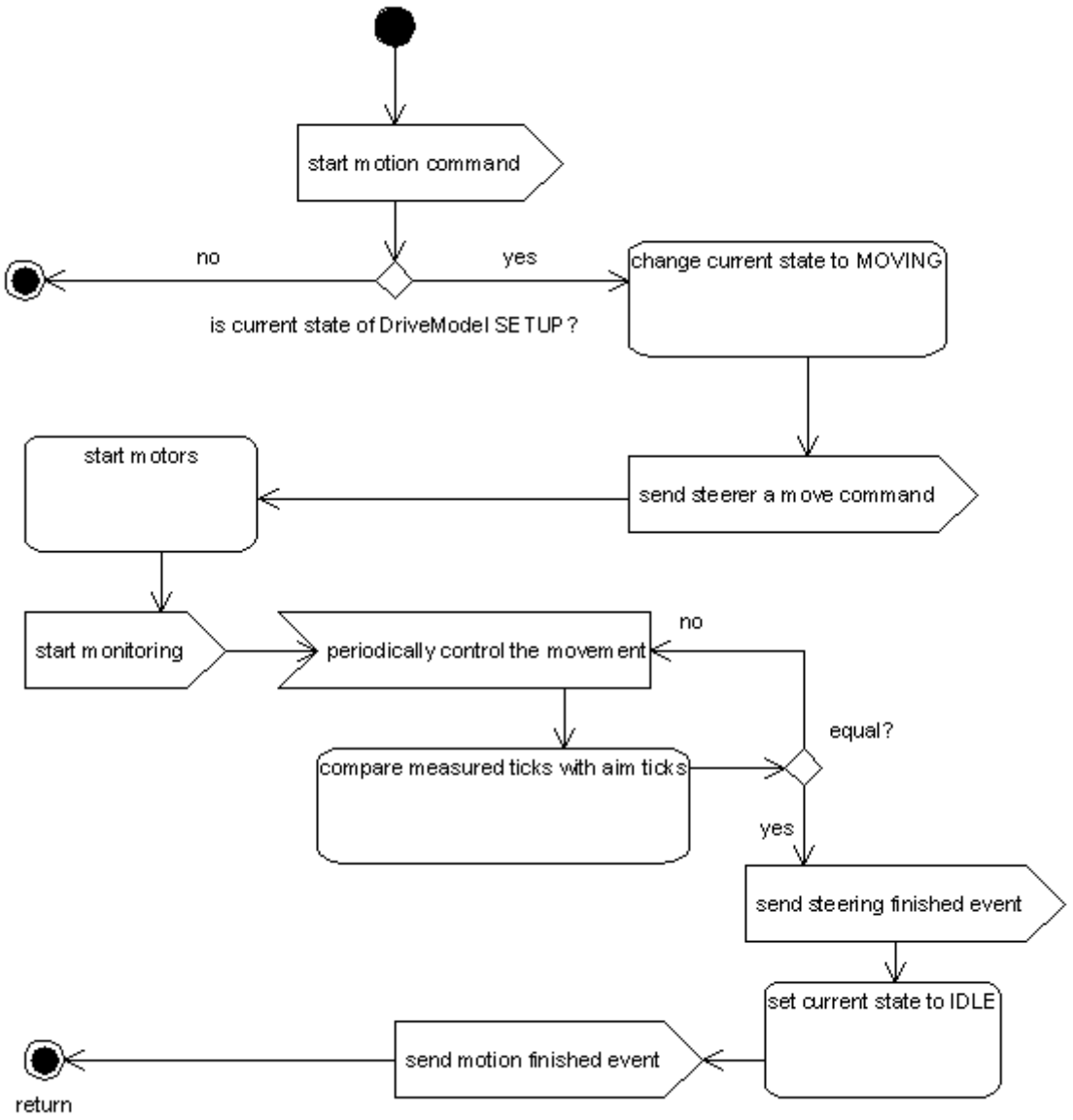
Schichtenmodell



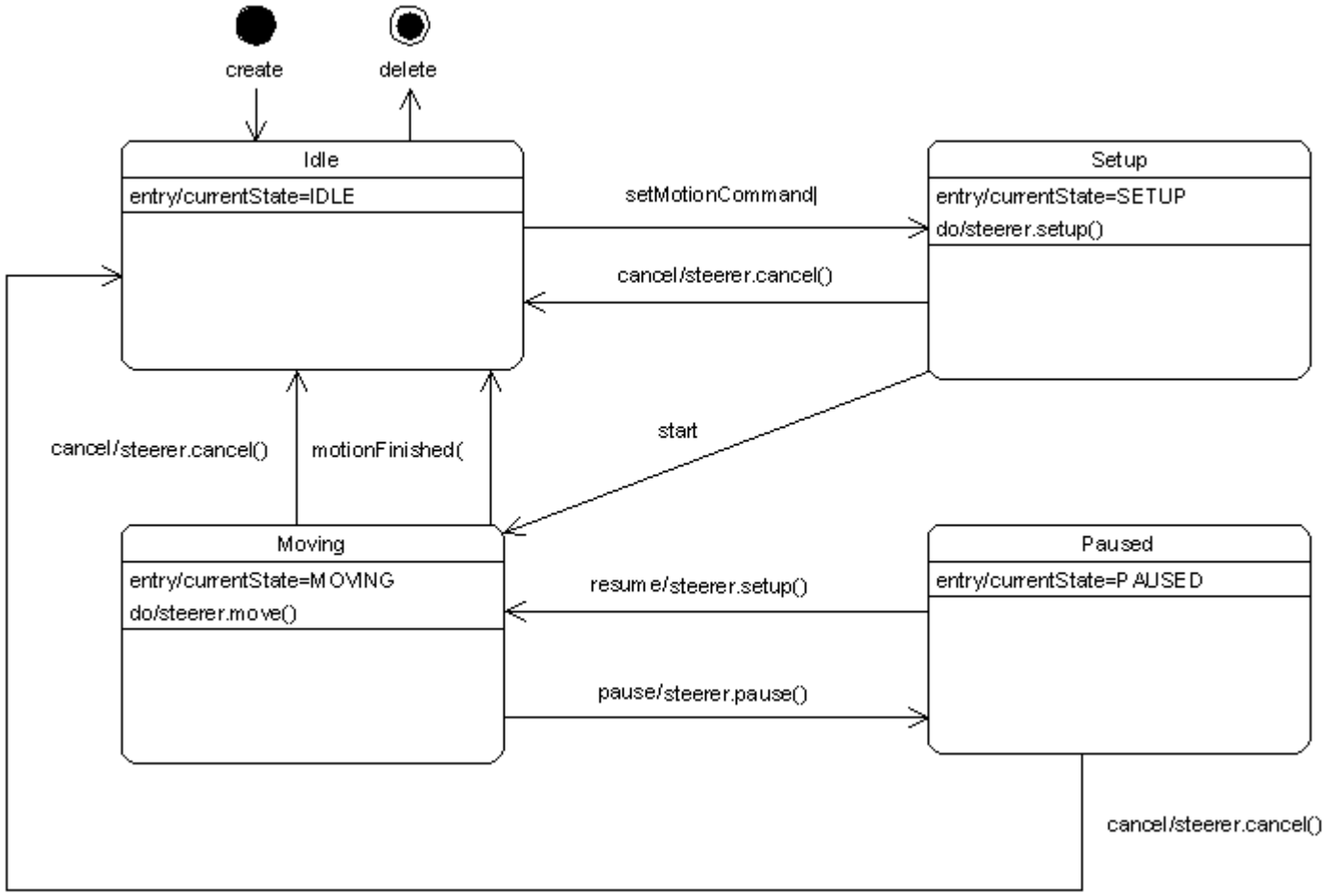
Anwendungsfälle

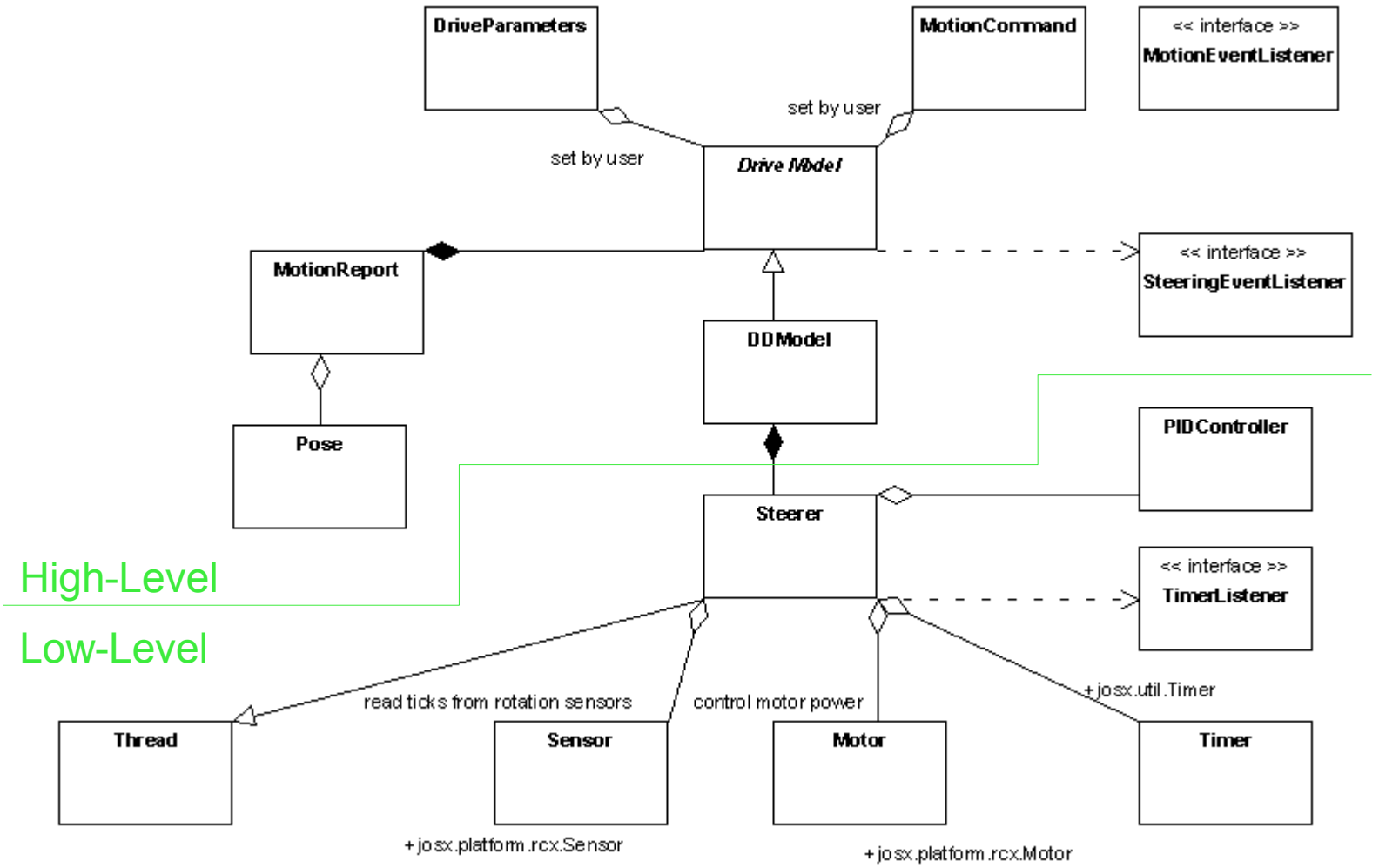


Fallbeispiel: start a motion



Zustände des DriveModel





LejON nutzen

- Anwendercode für ein Differential Drive (DIDI)

1. Initialisierung

1. DriveParameters setzen

```
// set dd parameters
DriveParameters params = new DriveParameters(
                                              16,   // ticks nominal
                                           10.5f,   // axle length
                                            7.0f,   // wheel diameter
                                  40.0f / 24.0f,   // gear ratio
                       DriveParameters.COUNT_FORWARD,   // count direction left
                       DriveParameters.COUNT_FORWARD); // count direction right
```

- Anwendercode für ein Differential Drive (DIDI)

1. Initialisierung
    1. DriveParameters setzen
    2. Motoren, Sensoren, Geschwindigkeitsvektor, PID-Parameter zuweisen


```
// set connection of motors
params.motorIdLeft          = Motor.C.getId();
params.motorIdRight         = Motor.A.getId();
// set connection of sensors
params.rotSensorIdLeft      = Sensor.S3.getId();
params.rotSensorIdRight     = Sensor.S1.getId();
// set velocities
params.velocities           = new int[] { 1, 3, 5, 5, 6, 6, 6, 6 };
// set values for pid controlling
params.maxMotorPower        = 7;
params.minMotorPower        = 1;
params.pidConstantKd        = 0.23;
params.pidConstantKi        = 0.3;
params.pidConstantKp        = 2.0;
params.timeBase             = 100;
params.maxTicksPerTimeBase = 7;
```

LejON nutzen

- Anwendercode für ein Differential Drive (DIDI)
  1. Initialisierung
     1. DriveParameters setzen
     2. Motoren und Sensoren zuweisen
     3. MotionEventListener für das DDModel zuweisen

```
// call the DDModel instance as a singleton
DriveModel myDrive = DDModel.instance( params );

// set MotionEventListener
myDrive.addMotionEventListener( this );
```

- Anwendercode für ein Differential Drive (DIDI)
  1. Initialisierung
     1. DriveParameters setzen
     2. Motoren und Sensoren zuweisen
     3. MotionEventListener für das DDModel zuweisen
  2. Bewegung auslösen
     1. MotionCommand erzeugen

```
// create a command for moving
MotionCommand cmd = new MotionCommand(
                        MotionCommand.TRANSLATE, // action
                                              4, // max. motor power
                                            100, // distance in [cm]
                                              0, // angle in [deg]
                                              0  // radius in [cm]
                                    );
```

LejON nutzen

- Anwendercode für ein Differential Drive (DIDI)
  1. Initialisierung
     1. DriveParameters setzen
     2. Motoren und Sensoren zuweisen
     3. MotionEventListener für das DDModel zuweisen
  2. Bewegung auslösen
     1. MotionCommand erzeugen
     2. MotionCommand an DDModel senden

```
// prepare moving
myDrive.sendMotionCommand( cmd );
```

LejON nutzen

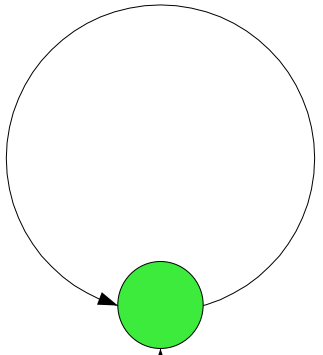
- Anwendercode für ein Differential Drive (DIDI)
  1. Initialisierung
     1. DriveParameters setzen
     2. Motoren und Sensoren zuweisen
     3. MotionEventListener für das DDModel zuweisen
  2. Bewegung auslösen
     1. MotionCommand erzeugen
     2. MotionCommand an DDModel senden
     3. Bewegung starten

```
// now move
myDrive.move();
```

LejON nutzen

- Anwendercode für ein Differential Drive (DIDI)
  1. Initialisierung
     1. DriveParameters setzen
     2. Motoren und Sensoren zuweisen
     3. MotionEventListener für das DDModel zuweisen
  2. Bewegung auslösen
     1. MotionCommand erzeugen
     2. MotionCommand an DDModel senden
     3. Bewegung starten

Evaluierung

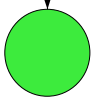


r = 50 cm

ABLAUF

1. Geradeausfahrt 250 cm
2. Drehung um 90° im UZS
3. Kreisfahrt mit Radius 50 cm
4. Drehung um 90° im UZS
5. Geradeausfahrt
6. Drehung um 180° gegen UZS

s = 250 cm



START/ZIEL

Testergebnisse

- Abweichung von Zielkoordinaten nach Abfahren vom Testparcours

	LejON		RotationNavigator	
Messung	Abweichung			
	X	Y	X	Y
1	-10	80	Nicht auswertbar, da Rotation um 180° erfolglos bleibt.	
2	-8	46		
3	-6	40		
4	-9	37		

Fazit

Zusammenfassung

- lejON
  - API zur Navigation zweirädriger Roboter mit LeJOS
  - Präziseres Fahren möglich durch PID-Regelung
  - Komfortable Nutzung für Anwender
  - Kapselung der Navigationsfunktionen
  - Implementierung für Single Differential Drives vorhanden (DIDI)
  - Erweiterbar für weitere Implementierungen, z. B. für Dual Differential Drives (HILDE)

Ausblick

- Nächste Mindstorms-Generation NXT
  - Erscheinungsdatum etwa Oktober 2006 (/3/)
  - 32-bit-MC
  - Motoren mit integrierten Rotationssensoren (1°-Auflösung!)
  - 100 Leistungsstufen für Motoren regelbar
  - LeJOS wird darauf portiert werden (/1/)
- Vorgehen zur Portierung von leJON
  - Low-Level an NXT anpassen
  - High-Level bleibt unverändert
  - Anwender definiert lediglich neue PID-Konstanten

Literatur

/1/ Andrews, P. et al.: leJOS, Java for the RCX; 2002; , http://lejos.sourceforge.net/; Stand: 22.03.2006

/2/ Baums, D.: LEGO MINDSTORMS Roboter, Der Profi Guide; Galileo Press GmbH, Bonn, 2000, ISBN 3-934358-39-X

/3/ McNally, M.: Lego Mindstorms NXT Key Product Features; The LEGO Group, Denmark, 2006; http://mindstorms.lego.com/press/2057/LEGO%20MINDSTORMS%20NXT%20Key%20Product%20Features.aspx; Stand: 22.03.2006

Vielen Dank fürs Zuhören!

Es folgt die Diskussion...

Folien und Quellen unter
http://lejon.florianruh.de/