Eine Fernsteuerung für den Lego EV3 Roboter

Von
Marius Christ

Zielsetzung

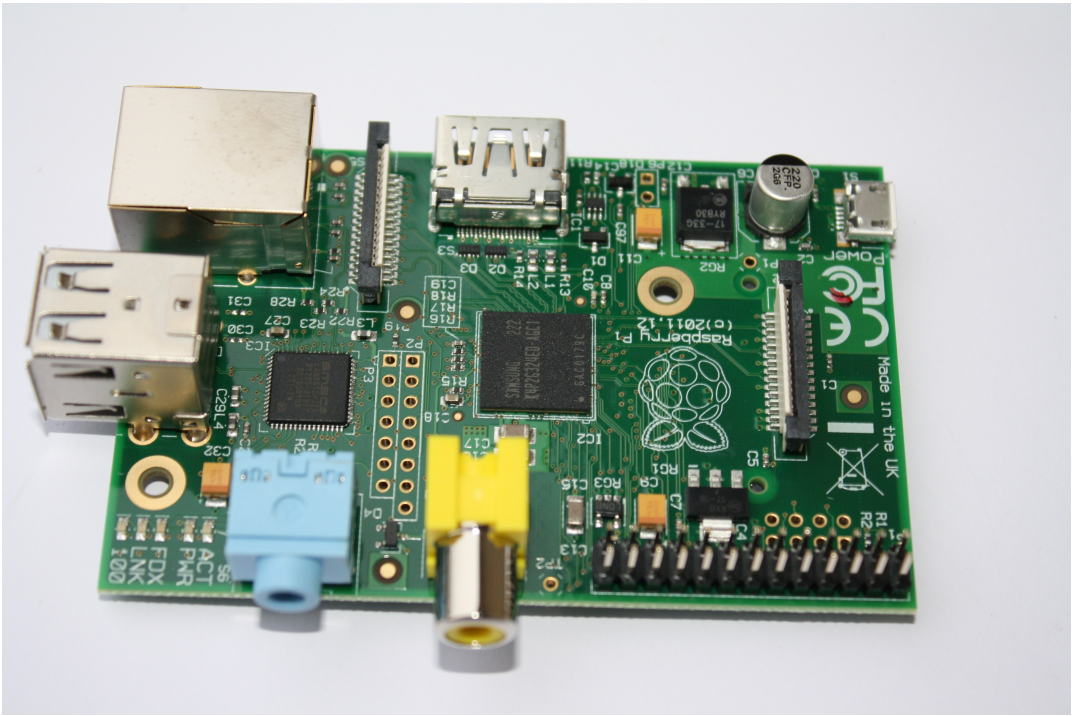
- Raspberry Pi
  - USB-Verbindung zum EV3
    - Direct Commands
  - Serverdienst
    - Steuerung des EV3 über USB
      - Byte Code
    - WLAN-Zugriff
    - Kamerastream
- Client
  - C# / WinForms
  - Netzwerkverbindung zu Raspberry Pi
  - Funktionen
    - Steuerung über Tastatur/Joystick
    - Wiedergabe Kamerabild

Lego EV3

- 3. Generation
  - August/September 2013
- 64MB RAM
- 300 MHz ARM-CPU
- MicroSD-Kartenslot
- USB-Hostport
  - WLAN-Adapter verfügbar
- Linux-Firmware
  - Erweiterbar über SD-Karte
    - ev3dev (Debian)
    - MonoBrick (C#)
    - LeJOS (Java)

Raspberry Pi

- 700 MHz ARM-CPU
  - 1GHz möglich
- 512MB RAM
- Kamera
- 2x USB 2.0
- 4x AA-Batterien
- SD-Karte als HDD
- Linux, BSD u.a.



Fernsteuerung

- Serielle Kommunikation
  - USB und Bluetooth möglich
    - USB-Verbindung sehr stabil
      - 500 Befehle/Sekunde problemlos
    - Bluetooth recht instabil
  - Systembefehle
    - Datei Up-/Download
    - Programm starten
    - Firmware aktualisieren
  - Direkte Befehle
    - Motorensteuerung
    - Sensordaten
    - Töne / LEDs / Display
  - Dokumentation kaum vorhanden
    - c_com.h u.a. im Quellcode der Firmware
    - EV3 API von BrickxCC

Beispiel: Bremsen

```
09 00 00 00 80 00 00 A3 00 06 01   Code
0  1  2  3  4  5  6  7  8  9  10   ByteNr
```

- Byte 0-1: Nachrichtenlänge
  - Little Endian
  - Nicht mitzählen!
- Byte 2-3: Nachrichtenzähler
  - Unwichtig
- Byte 4: Command Type
  - 0x80 == DIRECT_COMMAND_NO_REPLY
  - 0x00 == DIRECT_COMMAND_REPLY
  - Siehe c_com.h in den Firmwarequellen
- Byte 5-6: Anzahl der Variablen
  - Komprimiert (siehe c_com.h)
- Byte 7: opCode
  - A3 bedeutet opOutputStop
  - Siehe ev3_constants.h in der BricxCC API
- Byte 8: Daisy Chain Layer
  - Mehrere verbundene EV3
- Byte 9: Motor
  - Motor A ist das 1. Bit, Motor B das 2. Bit usw.
  - 00000010 0x02 Motor B
00000100 0x04 Motor C
00000110 0x06 Motor B|C
  - Motoren mit ODER verknüpfen
    - | Operator in C
- Byte 10: Bremsmodus
  - 00 ist Coast (kein Widerstand)
  - 01 ist Brake (Motoren halten still)

Beispiel: Fahren

```
0D 00 00 00 80 00 00 A5 00 06 81 00 A6 00 06
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
```

- Byte 7: opCode
  - A4 == opOutputPower
    - Asynchroner Motorenlauf
  - A5 == opOutputSpeed
    - Synchroner Motorenlauf
- Byte 8: Daisy Chain Layer
- Byte 9: Motoren
- Byte 10: Sinnloses Byte
  - Normal Speed-Angabe
    - Speed +/- 31 möglich
    - Byte 11 weglassen
  - Nicht dokumentiert
- Byte 11: Speed
  - Wertebereich +/- 100
- Byte 12: opCode
  - A6 == opOutputStart
  - Manuell notwendig
- Byte 13: Daisy Chain Layer
- Byte 14: Motoren

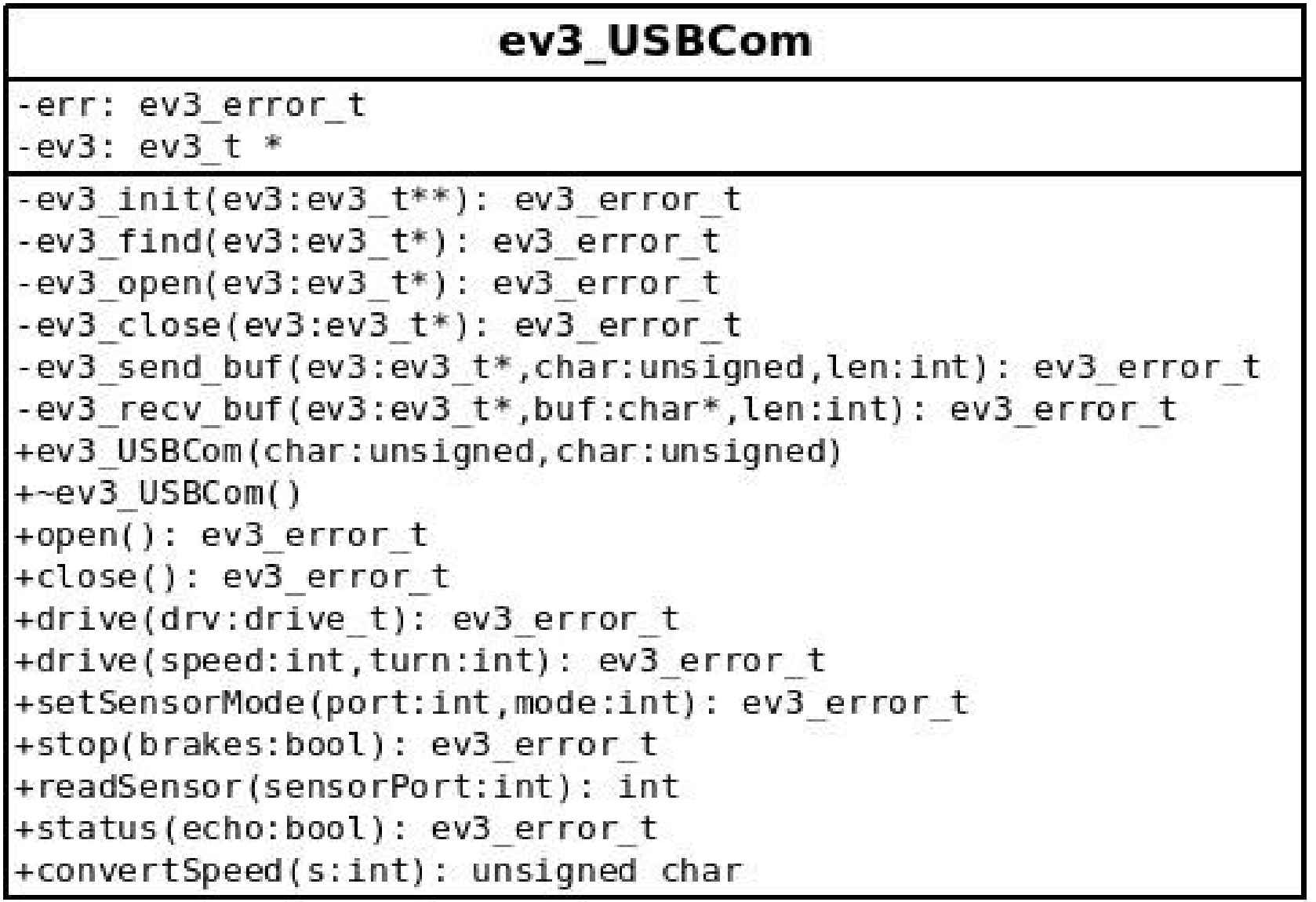
Struct drive_t

- Sammelt Fahrdaten in einer Struktur
  - Speed und Turn
  - Bremse, Bremsmodus, Lenkverhalten
- Wird zur Netzwerkkommunikation serialisiert
- Kann als Eingabe in die Methode ev3_USBCom::drive(drive_t drv) verwendet werden
  - Zentrale Auswertung und Ausführung

Klasse ev3_USBCom

- Verwaltet die Kommunikation mit dem EV3
  - USB-Variante
  - Bluetooth-Variante nicht gepflegt
- Enthält vorgefertigte Byte Codes
  - Byte Codes werden vor dem Senden angepasst
- Detaillierte Fehlercodes

UML-Diagramm: ev3_USBCom



Codebeispiele

oder Fragen?