Controlling LEGO®EV3 robots with Prolog

Sibylle Schwarz, Mario Wenzel
Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fakultät für Informatik, Mathematik und Naturwissenschaften
{sibylle.schwarz,mario.wenzel}@htwk-leipzig.de

22.10.2017

Motivation

- Robots are frequently used in introductory courses to robotics
- LEGO allows advanced contructions with complex behaviour and are used in university courses
- Robotics courses are sucessful in growing young peoples interest in STEM (ESF-funded project: “Roberta in Leipzig”)

Bindings to logic programming are rare. If we had them, we could:

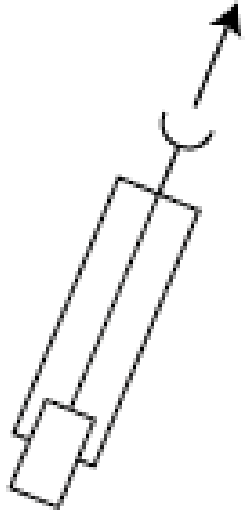
- experiment with intelligent control of autonomous robots and AI using logic programming
- connect existing introductory robotics courses with logic programming
- use robots in logic programming classes

Simple Robotics Experiments

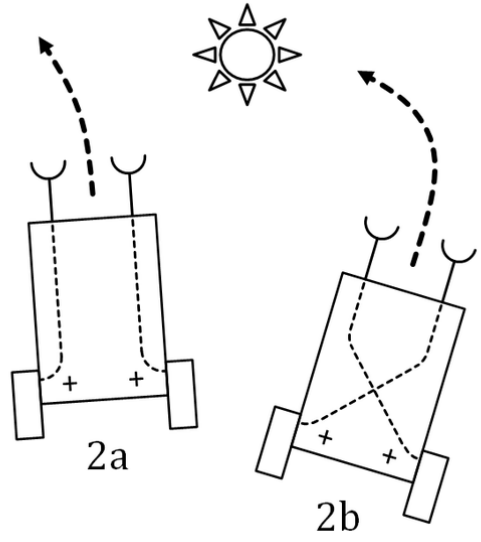
In our introductory robotics courses we use easy experiments that can be described by simple functions of the sensory inputs such as

- Obstacle avoidance
- Edge following
- Braitenberg vehicles (Demo)

B.-Vehicle 1 gets faster as it approaches a light source



B.-Vehicle 2 turns towards or from the light, depending on how the motors are connected



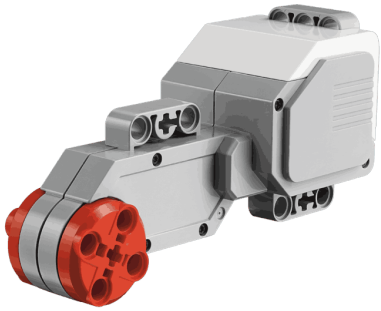
Related Work using LEGO Mindstorms Robots

Legolog: Inexpensive experiments in cognitive robotics (Levesque, Pagnucco, 2000, `http://www.cs.toronto.edu/cogrobo/Legolog`)

Prototype prolog api for mindstorms nxt (Nalepa 2008 - 31st Annual German Conference on AI, KI 2008)

EV3 – Hardware and Graphic Programming

EV3 brick

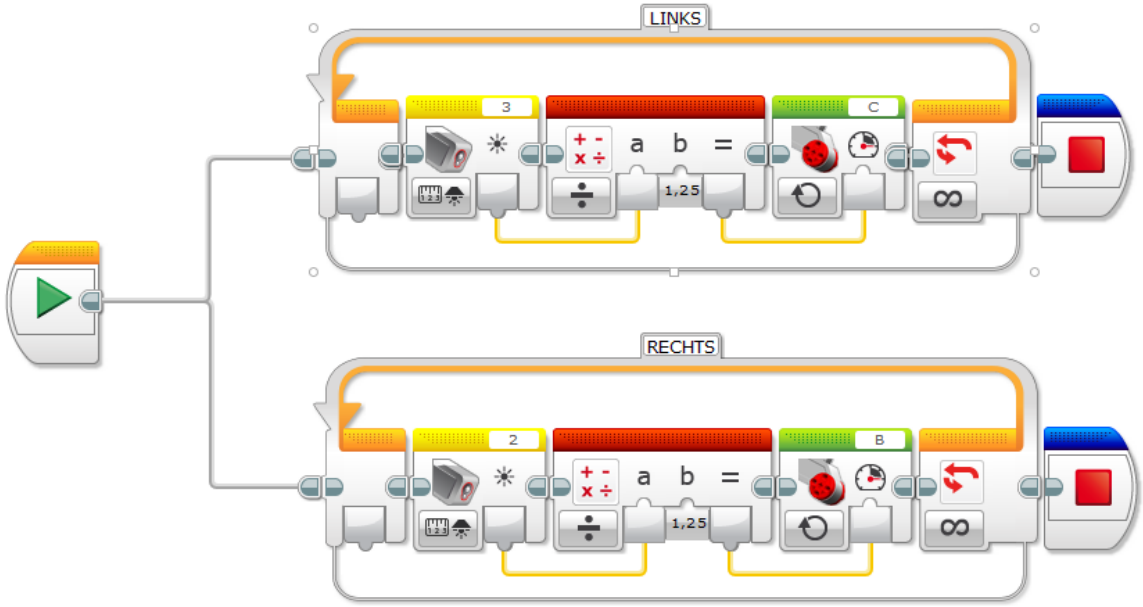


large motor

light sensor

ultrasonic sensor

LEGO RXC and NXT only had embedded firmware and a graphical programming environment.



ev3dev

- EV3 still has a graphical programming environment.
- but the EV3 brick is a proper computer
- there's a debian based OS for the EV3 brick (ev3dev)
- additionally there are bindings for
  Python, JS, Java, Go, C, C++, GObject (Vala)
- Virtual file system for the subsystems (sensors, motors)
  provided by a kernel module
  - directories correspond to subsystems and devices
  - readable files correspond to
    states (i.e. motor running or stalled) and
    data (i.e. read sensor values)
  - writable files correspond to
    commands (i.e. motor run forever) and
    target values (i.e. motor target speed)

Interaction with ev3dev Kernel Module

Processes (shell, language bindings) can interact with the robot and affect actions via the virtual filesystem
Motor directories: `/sys/class/tacho-motor/motor<n>/`
corresponding to motors detected during runtime

- the number *n* increases with newly or re-detected motors. The physical port of the device is in the file `/sys/class/tacho-motor/motor0/address`
- current speed is read from `/sys/class/tacho-motor/motor0/speed`
- target speed is written to `/sys/class/tacho-motor/motor0/speed_sp`
- to start the motor `run-forever` is written to `/sys/class/tacho-motor/motor0/mode`

Sensor directories `/sys/class/lego-sensor/sensor<n>/`
analogously

Implementation of Prolog Predicates for File Access

```
tacho_motor(Port, Type, Path) :-
  subsystem_detect(Port, Type, Path,
   '/sys/class/tacho-motor/motor*/').

speed_sp(Port, Speed) :-
 integer(Speed),
 max_speed(Motor, MaxSpeed),
 MaxSpeed >= Speed, Speed >= -MaxSpeed,
 speed_sp_file(Port, Filepath),
 file_write(Filepath, Speed).
```

Port	physical port of the EV3 brick (outA-outD)
Type	type of LEGO motor (large motor, medium motor)
Path	corresponding filesystem path ('/sys/class/tacho-motor/motor1/')
Filepath	virtual file to write to ('/sys/class/tacho-motor/motor1/speed_sp')

Prolog Predicates for Robot Control

To control the robot from Prolog we provide higher level predicates that abstract away the file access:

- connecting devices to ports
  `tacho_motor(Port, Devicetype)`
- read data and states (with side-effects)
  `col_ambient(Port, Lightlevel)`
- set and read commands and target values (with side-effects)
  `speed_sp(Port, Speed)`
- trigger complex actions
  `motor_run(Port, Speed)`, `motor_stop(Port)`, and
  `motor_run(Motor, Speed, Angle)`
- block computation until actions have resolved
  `motor_wait_while(Port, State)` and
  `motor_wait_until(Port, State)`

Example: Braitenberg Vehicles (Live Demo)

```
braitenberg2 :-
    col_ambient('in2', R), motor_run('outB', R),
    col_ambient('in3', L), motor_run('outC', L),
    braitenberg2.

motor_run(Motor, Speed) :-
    tacho_motor(Motor), speed_sp(Motor, Speed),
    command(Motor, 'run-forever').

col_ambient(Port, Val) :-
    light_sensor(Port), mode(Port, 'COL-AMBIENT'),
    value(Port, Val).

value(Port, Value) :-
  value_file(Port, Valuefile),
  file_read(Valuefile, Value).
```

Results

- We created a basic Prolog Interface for LEGO EV3 robots.
- That is idiomatic Prolog (as far as possible).
- Sensors and actors of the robot can be operated from Prolog programs.
- LEGO EV3 robots can be controlled by Prolog programs using the presented bindings.
- We created various simple introductory experiments (Braitenberg vehicles, obstacle avoidance).

Further Work and Next Steps

Use Prolog features for

- path planning
- maze experiments
- trajectory control
- autonomous driving

Extend the API for more devices and other device types:

- Support more sensors
- Display control (it's just a Framebuffer device)
- Audio output (ALSA works fine)
- Introduce a robot configuration for higher level control (such as go 30 cm, turn 60 degree right)
- Integrate a Prolog launcher into the ev3dev GUI