Workshop „Swarm robotics“

Reactive control of LEGO robots with MDL2e

Thomas Schamm

23.05.2006
A look on the structure of the presentation

- Project SIMON
- Behaviour based robot control
- LEGO robot – pros and cons
- IPAQs as extensions for LEGO robots
- The programming language NQC
- Behaviour based control with MDL2e
- User-friendly input through the MDL Editor
- The interface between IPAQ and RCX
- Conclusion and future prospects
Introducing a development environment for SO-Software

- **Project SIMON**
  - Developing secure, self-organizing software for mobile components in factory automation

- **Analyzing the effect of self-organization on requirements like**
  - Security
  - Real-time capability
  - Error tolerance
  - Efficiency
  - User-friendly Interaction

- **Questioning of the reduction of development effort with principals of organic computing**
Behaviour based controlling of a LEGO robot

- LEGO robot out of the box:
  - Simple behaviour based control possible (see Figure 1)
  - Complicated interaction between multiple robots
  - Follows a given algorithm

- Goal:
  - Multiple interacting robots
  - Failsafe communication
  - Task change if necessary

```c
#define MOTOR_L OUT_A
#define MOTOR_R OUT_B
#define SNSOR_L SNSOR_1
#define SNSOR_R SNSOR_2

void task_main()
{
  SetSensor(SENSOR_L + SENSOR_R, SENSOR_TOUCH);
  SetPower(MOTOR_L+MOTOR_R, 3);
  OnFwd(MOTOR_L+MOTOR_R); // Turn on Left and Right Motor

  while (true) // Go Forward until obstacle
  {
    if (SENSOR_L) // Left Touch Sensor pressed
    {
      OnRev(MOTOR_R); // Turn to the right
      Wait(2);
      OnFwd(MOTOR_R); // Go Forward
    }
    else if (SNSOR_R) // Right Touch Sensor pressed
    {
      OnRev(MOTOR_L); // Turn to the left
      Wait(2);
      OnFwd(MOTOR_L); // Go Forward
    }
  }
}
```

Figure 1: Simple NQC Program
Using a LEGO robot as development platform

**Advantages:**
- Easy robot construction for almost every problem
- RCX programmable in C, Java and other languages
- Simple actors and sensors
- Cheap in comparison to other development platforms

**Disadvantages:**
- Only 6 Kbyte space for user programs on RCX
- Available sensors leave robot almost blind (only touch and light sensors)
- No communication except IR
- Manually reprogramming of RCX in case of task change
Eliminating the cons by linking the RCX with an IPAQ

- The IPAQ 5550 as communication and control platform
  - Interaction with RCX over serial interface
  - Wireless communication:
    - 802.11b Wi-Fi
    - Bluetooth v.1.1
    - IrDA
  - Possibility of running complex behaviour based programs
    → MDL2e
  - Graphical user interface possible

Figure 2: Behaviour based control of the LEGO robot
NQC doing basic robot control

- **NQC as programming language for LEGO robot**
  - Textual replacement of the graphical programming language RIS
  - Support for control of motors and sensors
  - Communication with other RCX over IR only

- **NQC useful for**
  - Checking sensor values and storing them
  - Calling motor commands for navigation of the LEGO robot
  - Making sensor values and motor commands available for MDL2e
MDL2e in charge of reactive control

- Useful to control several kinds of units (e.g. Jasmin robots)
- Job assignment stored in XML file
  - XML file generated with MDL Editor
  - Plan contains reactive control sequence
- MDL2e frequently communicates with RCX
  - Every step a RCX command is called
    - e.g. check sensor value
    - e.g. drive forward
  - Runs until end of XML file or new XML file is called
- Communication takes place over developed serial protocol

Figure 3: Workflow sequence
Creating jobs using the GUI

MDL Editor

Figure 4: MDL Editor creating XML file for obstacle avoidance
Creating jobs using the GUI

- **MDL Editor created XML file**

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE AuftragScript PUBLIC 'AuftragScriptSchema.dtd'
 'AuftragScriptSchema.dtd'>

<AuftragScript>
  <MISSION id="AVOID_OBSTACLES">
    <EXEPATH id="LEGO_ROBOT" timeout="infinite">
      <INSTRUCTION id="SET_SENS_MODE" timeout="1"
      arg0="50" arg1="2" arg2="1" />
      <JOB id="GO_FORWARD" interrupt="SENS_TOUCHED">
        <INSTRUCTION id="GO_FORWARD" timeout="1" arg0="33" arg1="131" />
        <INSTRUCTION id="WAIT_FOR_SENS" timeout="infinite" />
      </JOB>
      <JOB id="TURN" interrupt="TRUE">
        <INSTRUCTION id="TURN" timeout="10" arg0="225" arg1="65" />
      </JOB>
    </EXEPATH>
  </MISSION>
</AuftragScript>
```

Figure 5: XML file for obstacle avoidance
Communication between MDL2e and RCX

- **Development of communication interface** `RCXCommObj`
  - Based on existing serial communication tool `send.c` [1]
  - Correct packet must look like:
    
    \[0x55 \ 0xff \ 0x00 \ D1 \ ~D1 \ D2 \ ~D2 \ \ldots \ Dn \ ~Dn \ C \ ~C\]
  - Packet header is `0x55 \ 0xff \ 0x00`
  - Bytes `D1`, `D2`, ..., `Dn` contain opcode
  - `RCXCommObj` transfers opcodes and receives the RCX answer

- `RCXCommObj` is called through MDL2e and sends a specified opcode to the RCX
- The RCX answer is returned to MDL2e for further processing

Two class methods must be implemented

```
<JOB id="GET_SENS_VALUE" interrupt="SENS_TOUCHED">
    <INSTRUCTION id="GET_SENS_VALUE" timeout="10"
        arg0="18" arg1="9" arg2="1" />
</JOB>
```

Figure 6: Example XML element

**myExecutable::execute()**

- Class method is called if XML element INSTRUCTION contains XML arguments arg0, ..., argN
- Class method calls RCXCommObj for transfer of arguments
- Figure 6 shows example argument
Implementing communication interface in MDL2e

Two class methods must be implemented

```xml
<JOB id="GET_SENS_VALUE" interrupt="SENS_TOUCED">
  <INSTRUCTION id="GET_SENS_VALUE" timeout="10"
    arg0="18" arg1="9" arg2="1" />
</JOB>
```

Figure 6: Example XML element

myInterrupt::evaluate()

- Class method is called every time step to check whether XML argument
  interrupt is true or false
- Check is done by calling again the RCXCommObj
- Execution of XML element is canceled if either interrupt is true or
  timeout counter is 0
Conclusions of this work

- IPAQ extends RCX for communications and computation issues
- NQC necessary for basic control of the RCX
- MDL2e as extended reactive control for robots
- MDL Editor as GUI to create jobs as XML files
- Implementation of `myExecutable` and `myInterrupt` for interlinking MDL2e with the RCX
- Interlink is done via the `RCXCommObj`
Future prospects

- MDL2e serves as reactive control for all units of a factory
  - e.g. transportation robot
  - e.g. production unit

- Every unit uses its own MDL2e, tasks are communicated as XML files

- Distributed automatic control of production

→ Project SIMON: MDL2e serves as tool for reduction of development effort
Workshop „Swarm robotics“
Reactive control of LEGO robots with MDL2e

Thomas Schamm