The Line-Following LEGO Robot

and programming the LEGO Mindstorms in general

Fred Barnes, Systems Research Group
Computing Laboratory, University of Kent
F.R.M.Barnes@kent.ac.uk
Lego Programming

This is fun! :-)

Uses the Lego Mindstorms “brick”

Can drive up to 3 motors and respond to 3 sensor inputs
  - sensors include: touch, light, rotation
  - also has an IR port

The brick contains:
  - a small 8-bit RISC architecture chip (16 MHz)
  - some memory: 32 kb of RAM, 16 kb of ROM
  - interface gadgetry (inputs, outputs, IR, buttons, display), and batteries
Programming the brick:

- humans like abstract (or high-level) program code
- not the low-level code the H8 processor actually executes

Diagram:

- NQC/mindstorms
- Java
- occam

- RCX byte-code
- Java byte-code
- ETC byte-code

- RCX interpreter
- lejos JVM
- transterpreter

- Firmware (provided by Lego)
- Hitachi H8 RISC microprocessor
- sensors/motors

32k RAM
16k ROM
The firmware (software that is built into hardware) provides access to the device’s inputs and outputs
- also (initially) deals with downloading code via the IR port

The various “interpreter” layers are where the magic happens
- “RCX interpreter” runs RCX byte-code in the device
- “lejos JVM” provides a (very basic) Java Virtual Machine that executes the Java byte-code
- “transterpreter” runs ETC byte-code in the device

The choice of language is up to the programmer — who only needs to deal with NQC, mindstorms, Java or occam (not any of the low-level stuff!)
NQC is a C-like language (not quite C):

```c
#include "NQC.h"

task main ()
{
    SetPower (OUT_C, 3);
    SetPower (OUT_A, 3);
    ClearTimer (0);
    OnFwd (OUT_A + OUT_B);
    while (Timer(0) < 100) {
        /* do nothing! */
    }
    SetPower (OUT_C, 2);
    while (Timer(0) < 200) {
        /* do nothing! */
    }
    Off (OUT_A + OUT_B);
}
```

- set motor power
- clear timer 0
- turn motors on
- wait a while
- set motor power
- wait a while
- turn motors off
Java uses a special ‘brick’ API:

```java
public class BrickTest {
    private static int x = 0;
    private static class MyTimer implements TimerListener {
        public void timedOut () {
            if (x == 0) {
                Motor.C.setPower (2);
            } else {
                Motor.A.stop ();
                Motor.C.stop ();
            }
            x = x + 1;
        }
    }
}
```
And the programming is slightly different:

```java
public static void main ()
{
    Motor.C.setPower(3);
    Motor.A.setPower(3);
    TimerListener tl = new MyTimer();
    Timer tim = new Timer(1000, tl);
    Motor.A.forward();
    Motor.C.forward();
    tim.start();
    // now do nothing forever (!)
    for (;;);
}
```

the first time the timer goes off, it changes the motor power (speed)

every time after that, if just turns the motors off
This robot attempts to follow a line drawn on the ground underneath it:

- two motors: left and right
- one sensor: light

Basic operation is this:

(start) → go forwards → on-line? (yes) → go forwards → search for line

on-line? (no) → on-line? (yes) → on-line? (no) → continue for a bit then stop

Searching for the line is the hard part ...
Finding the line is done by turning the robot left and right (looking)

- physical configuration of this robot helps :-)

\[ \text{\begin{align*}
\text{(start)} & \quad t = 1 \\
\text{reset timer} & \quad \text{turn left} \\
\text{timeout ?} & \quad \text{on-line ?} \\
\text{yes} & \quad \text{no} \\
\text{t = t + 1} & \quad \text{turn right} \\
\text{on-line ?} & \quad \text{reset timer} \\
\text{yes} & \quad \text{no} \\
\text{t = t + 1} & \quad \text{timeout ?} \\
\text{yes} & \quad \text{no} \\
\text{t = 7 ?} & \quad \text{random search} \\
\text{yes} & \quad \text{no} \\
\text{(found line, return)} &
\end{align*}} \]