Real-Time Systems Programming

Summer-Semester 2002
Lecture 12
24 May 2002

Introduction to Mindstorms

Overview

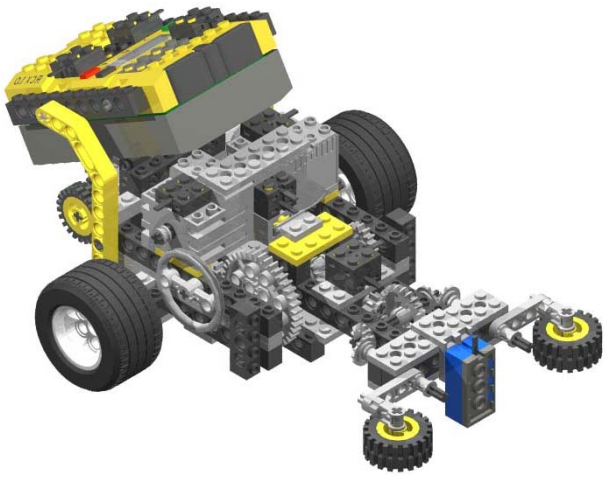
1) The robotics command system (RCX)
2) Family history – Media Lab's programmable bricks
3) The RIS standard programming environment
4) Alternative developments
5) legOS

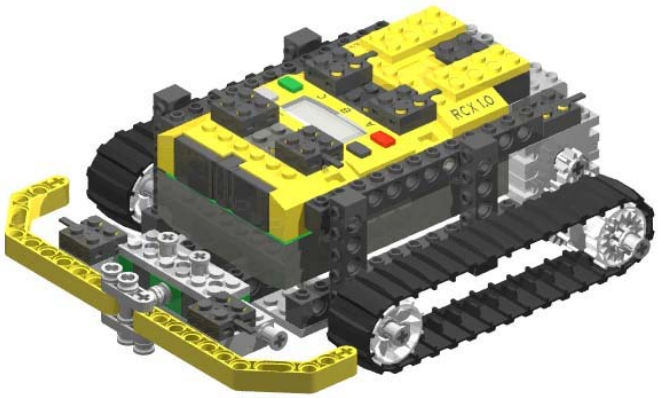
Recall: What is a Real-Time System ?

- ***A definition:*** ***Real-time systems (RT-Systems)*** are those computational systems that
  - offer an assurance of timeliness of service provision
- ***Another definition:*** ***RT-systems*** are those where the correctness of the system behavior depends
  - on the logical results of the computations, ***and also***
  - on the physical time when these results are produced
- ***Yet another definition:*** ***RT-systems*** are those that
  - have to be designed according to the dynamics of a physical process

Lego Mindstorms – Introduction

- ***Mindstorms***: technically most advanced ***Lego*** brand
- Mindstorms ***Robotics Invention System*** (***RIS***): allows construction and programming of various robots





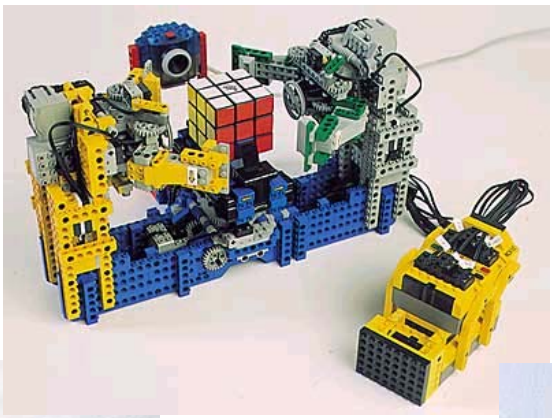
Knudsen/O'Reilly

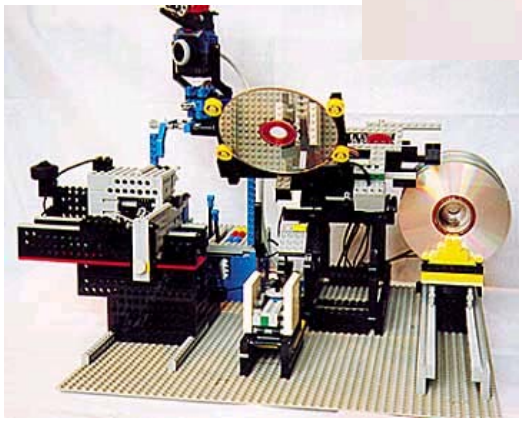
- The main applications are autonomous vehicles – but numerous other machines have been built as well: card dealers, copy machine, Mars rover, walker, fire extinguisher, ...

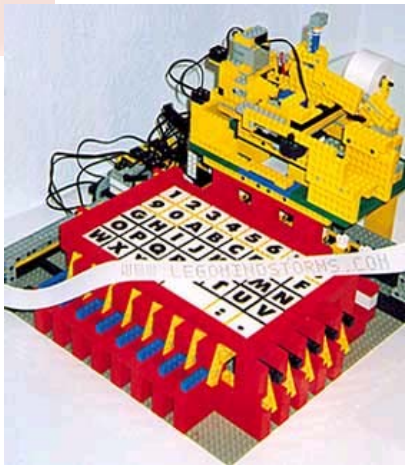
Robot ≠ Robot

IEEE Spectrum

Are These Still Robots?







IEEE Spectrum

The Robotics Command System

- The core of the RIS is the ***Robotics Command System*** (***RCX***)

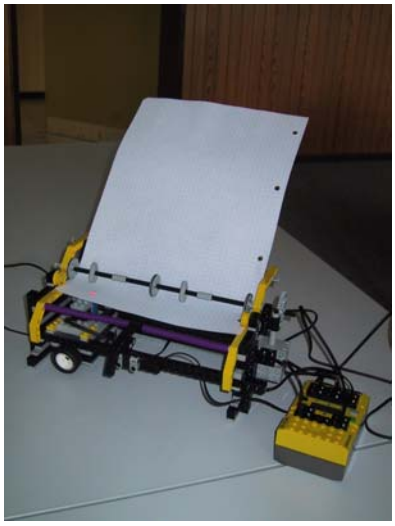
- Hitachi H8 Microcontroller
- 3 sensor inputs
  - Touch sensor
  - Light sensor
  - Rotational sensor
- 3 motor outputs
- Infrared port
- Speaker

The RCX – A Typical Real-Time System!

- The RCX is a typical real-time system and has to incorporate ***time*** in its actions on various scales
- ***Macroscopic real-time aspects:***
  - The duration of *motor-on* signals and the ***positioning*** of the robot are directly interrelated
  - Most meaningful robotic ***strategies*** have to make use of delays and time-outs at some point
- ***Microscopic real-time aspects:***
  - The motors are controlled using ***pulse-width modulation***
  - The ***speaker*** must be able to generate various frequencies

Example: Scanner

- ***RT Problem:*** Correlation sensor data – pixel coordinates



The RCX – A Typical Embedded System!

RCX is controlled by a digital CPU ...

... but still quite different from a non-embedded computer

	RCX	**Laptop**
Applications	Robot Control	"General Purpose" Office applications, SW development, ...
Cost	€ 220,- Includes SW and building materials for a robot	€ 2500,- Barely any SW, no peripherals
Unit count	100 000s *(?)*	1000s *(?)*
Size	6.5 x 3.5 x 9.5 = 216 cm³	31.0 x 3.5 x 26.5 = 2875 cm³
Weight	220 g	2850 g
Power	6 AA Batteries	16V, 4.5A (72W) transformer
SW Updates	To be avoided	No problem (*ahem* ...)

RCX vs. Laptop – Hardware Comparison

- Differences in ***requirements*** ⇒ differences in ***design***

	RCX	**Laptop**
CPU	Hitachi H8 (8-Bit microcontroller)	Pentium III (32-Bit microprocessor)
Speed	16 MHz	800 MHz
RAM	32 KB	192 MB
ROM	16 KB	192 KB (?)
Addl. storage	None	20 GB harddrive
Display	1 x 3 cm² LCD	28.5 x 21.5 cm² = 613 cm² TFT
Keyboard	4 buttons	94 buttons + mouse
Further Interfaces	3 sensors, 3 actuators IR port, speaker	USB, serial, parallel, Ethernet, modem, PCMCIA, int./ext. speakers, int./ext. mike, ext. monitor, ext. keyboard/ mouse, CD/DVD, floppy, IR Port

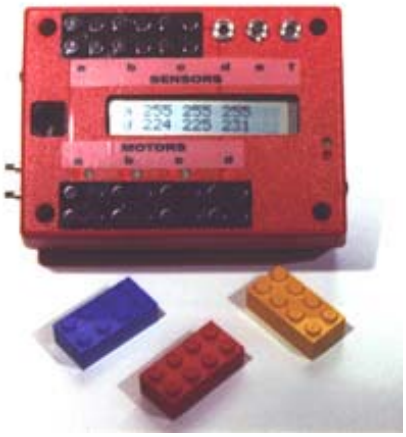
Mindstorms Family History

- Mindstorms Roboter: strongly influenced by
  - ***Programmable Brick*** project
  - at the ***Media Lab*** of the Massachussets Institute of Technology (MIT)
- 1971: "*Twenty Things to Do with a Computer*"
- 1987 to 1989: ***The 6502 Programmable Brick***
  - Focus on children, ran a LOGO interpreter
- 1989 to 1991: ***Electronic Bricks***
  - Attempt to provide a concrete paradigm for "wiring" behaviors rather than programming them

- "*Twenty Things to Do with a Computer*", by Seymour Papert and Cynthia Solomon was a visionary paper that already outlined many concepts that reached reality much later

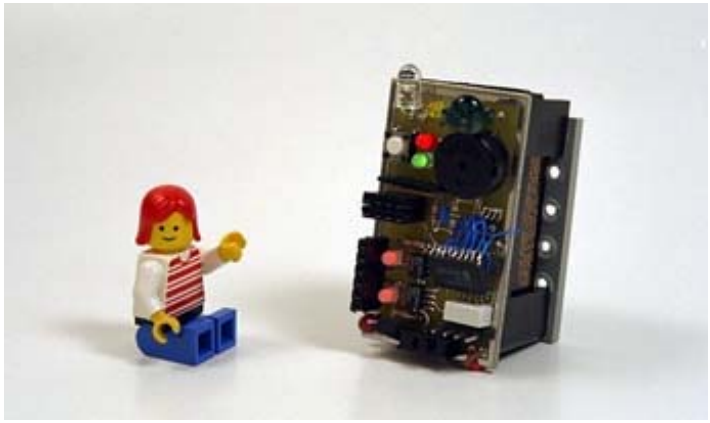
More Programmable Bricks

- 1993 to 1995: The ***Pocket Programmable Brick***
  - Very compact
  - Had IR port, 8 sensor ports, 4 motor ports, sound I/O
  - Runs ***Brick Logo***
- 1994 to present: The ***Model 120 Programmable Brick***
  - More economical version of the Pocket Programmable Brick
  - Look and feel of a commercial product
- 1998: ***Lego Mindstorms***



The Next Generation: Crickets

- ***Cricket***
  - Also from MIT Media Lab
  - Cross-breeding between ***Programmable Brick*** and wearable ***Thinking Tag***
  - Powered by 9V battery
  - 2 actuators (motors)
  - 2 sensors
  - IR communication
- ***Example Applications:***
  - Robotic
  - Body monitoring
  - Data collection



Designing a Robot with the RIS

Design of a Mindstorms robot consists of two steps:

- ***Building the robot***
  - An art in itself
  - Numerous texts on mechanical ***robot design*** in general and the ***mechanics of Lego bricks*** in particular
  - Not be the focus of this class
- ***Programming the robot***
  - Will concentrate on this step
  - RIS comes with a programming environment ...
  - ... and the ***Mindstorms user community*** has also developed several alternatives by now

The RIS Programming Environment

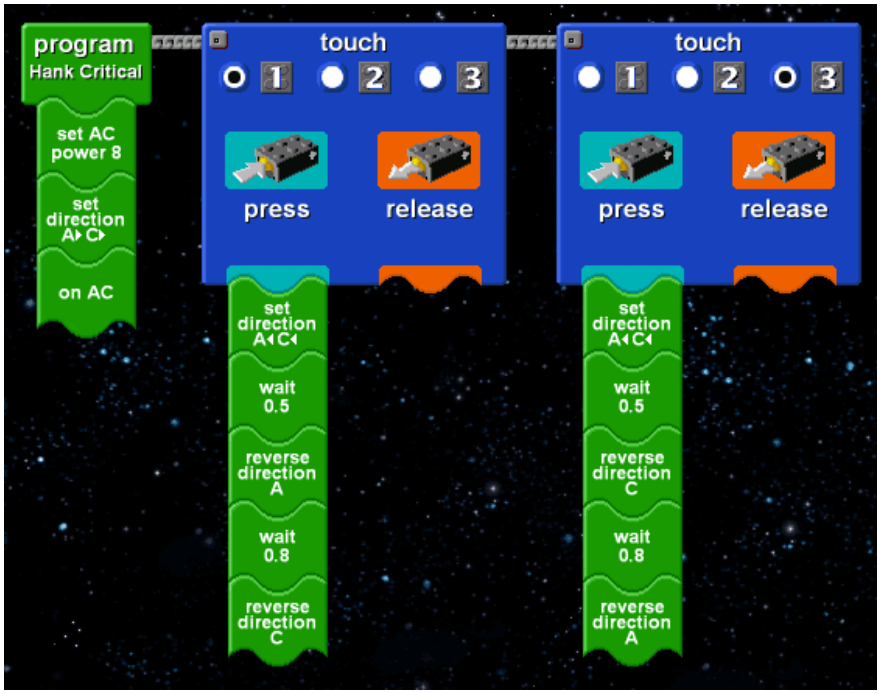
- Consumer profile (intended) of Mindstorms brand:
  - Aged 12+
  - ***No prior programming experience***
- RIS comes with graphical programming environment
- ***Cross Development***:
  - Programs developed on a Windows platform
  - Then download (via the IR port) to the RCX
- RIS programming environment
  - Produces ***byte code***
  - This then interpreted by ***firmware*** on the RCX

The RIS Programming Environment

- On the ***development platform*** (***Windows***), there is
  - RCX code (***Brick*** language)
  - Spirit.ocx – an ActiveX-Control on which the Brick language development system is based
- Byte code transferred to the RCX via
  - Serial port of the PC + IR transmitter
- On the ***run-time platform*** (the RCX)
  - ***RAM:*** Byte code + Firmware
  - ***ROM:*** Boot firmware

The Brick Language

- The graphical language consists of ***bricks*** of different color (this is Lego, after all ...)
- Green:
  Commands
- Blue:
  Sensor Watchers
- Red:
  Control Blocks
- Yellow:
  Macro Blocks



The Brick Language

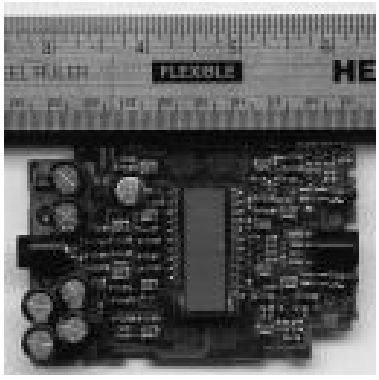
- In principle, the language
  - has all there is needed for programming a robot
  - including some (from a classical CS-point of view) non-standard features
- ***Concurrency***
  - ***Note:*** if the trigger for a task that is already running is activated again, the task is ***re-started immediately***
- ***Sensor/actuator control***
- ***Delay and time-out primitives***

Limitations of the Brick Language

- No variables – just a simple counter
- No expressions
- No function calls
- No nesting
- Incomplete control of machine state
- No concept of protected resources or atomic actions

Cracking the RCX

- Mindstorms users have undertaken extensive ***reverse-engineering efforts*** of the RCX shortly after they hit the market in early September 1998
- By Oct. '98, Kekoa Proudfoot (Stanford U) had reverse-engineered the RCX opcodes, thus opening the door for
  - ***Low level programming***
  - Additional ***tools***
  - ***Non-Windows*** development platforms
  - The ultimate: ***firmware replacement*** (no byte code no more!)



NQC – Not Quite C

- ***NQC*** replaces the RIS development system
- Closely resembles C
- Code is compiled and downloaded
- As it does not use spirit.ocx anymore, it is not restricted to Windows – also runs under MacOS and Linux
- There also exists a windows-based interface to this, ***RcxCC***
  - Editor with syntax-highlighting
  - Real-time control of the RCX

Example in NQC: Light Watcher

- Control Software of a robot that follows a line
  - Uses the light sensor

```
int state;

#define LEFT 0
#define RIGHT 1

#define DARK2 35
#define LIGHT2 40

#define POWER 7

#define TIMEOUT 50
```

```
task main() {
  state = LEFT;
  SetSensor(SENSOR_2, SENSOR_LIGHT);
  SetPower(OUT_A + OUT_C, POWER);
  start lightWatcher;

  while (true) {
    if (SENSOR_2 < DARK2)
      OnFwd(OUT_A + OUT_C);
  }
}
```

Light Watcher II

- **`sub:`** denotes a subroutine

```
task lightWatcher() {
  while (true) {
    if (SENSOR_2 > LIGHT2) {
      toggle();
      Wait(TIMEOUT);
      if (SENSOR_2 > LIGHT2) {
        toggle();
        Wait(TIMEOUT * 2);
      }
    }
  }
}
```

```
sub toggle() {
  if (state == LEFT) {
    OnRev(OUT_A);
    OnFwd(OUT_C);
    state = RIGHT;
  }
  else {
    OnFwd(OUT_A);
    OnRev(OUT_C);
    state = LEFT;
  }
}
```

Limitations of the Original Firmware

- NQC still produces original ***byte code***
- Programs written in NQC therefore have the same, firmware-induced limitations:
  - Only 32 variables
  - Limited hardware and display control
  - No real process synchronization
  - At most 10 simultaneous tasks
  - Subroutines: no nesting, no parameters, max 8 per program
- ***To overcome these restrictions:*** Firmware Replacement

pbFORTH – A Firmware Replacement

pbFORTH is a Forth dialect

- Stack-based
- Commands are interpreted interactively
- One programs by successively extending a dictionary
- There is a set of pre-defined words that manipulate the stack: **`DUP`**, **`OVER`**, **`PICK`**, **`SWAP`**, **`ROT`**, **`DROP`**, ...
- There are also words for control flow, mathematical operations, etc.

pbFORTH Capabilities

- pbFORTH contains words for controlling sensors and actuators, and the LCD display
- Timers:
  - As in the Brick language, four 1/10-sec timers
  - In addition, ten 1/100-sec timers
- Functions for power management:
  - **`POWER_OFF`**, **`POWER_GET`**
- Cooperative (instead of preemptive) multi-tasking

A Thermometer in pbFORTH

```
HEX

: buttonState RCX_BUTTON DUP BUTTON_GET @ ;

: isRunButtonPressed buttonState 1 AND ;

: showTemperature
  3003 SWAP 3001 LCD_NUMBER
  LCD_REFRESH ;

: clear LCD_CLEAR LCD_REFRESH ;
```

```
: thermometer
  RCX_INIT SENSOR_INIT BUTTON_INIT
  2 1 SENSOR_TYPE
  A0 1 SENSOR_MODE
  BEGIN
    BEGIN
      1 SENSOR_READ 0=
    UNTIL
    1 SENSOR_VALUE
    showTemperature
    isRunButtonPressed
  UNTIL
  clear ;
```

The pbFORTH Design Flow

- On the ***development platform*** (***Windows, Linux, MacOS, ...***):
  - Terminal emulator
- Transfer byte code via the serial port of the PC and the IR transmitter
- On the ***run-time platform*** (the RCX):
  - in RAM: Application code + pbFORTH interpreter
  - in ROM: Boot firmware

legOS – Another Firmware Replacement

- ***legOS***: a firmware replacement
  - Developed by Markus Noga (TU Karlsruhe) and others
  - Provides basic ***POSIX*** functionality
    - Process management
    - Event notification
    - Counting semaphores
  - Allows running the RCX in ***native mode***
  - Can use full 32 KB memory (remember – this is the embedded world)
  - Development platform does not have to be ***Windows*** – can be ***Linux***, for example

Developing for legOS

- Code running on ***legOS*** can be compiled using **`gcc`**
  - There exists a Hitachi H8 backend for **`gcc`**
  - Can make full use of the C language – arrays, pointers, memory allocation, ...
  - No limit of 32 variables
- There are also simulators for legOS
  - emulegOS
  - legosim
  - ***Warning***: usually the simulators are a couple of revisions behind legOS itself

Parts of a legOS Program

- A program running on top of legOS consists of Hitachi-H8 machine code and several data areas
- The parts of an RCX binary are:
  - `.text`: program code; also contains initialized, constant data
  - `.data`: initialized, variable data
  - `.bss`: non-initialized, variable data
  - `.stack`: run-time stack for local variables, return addresses etc.

Interpreted vs. Native Code

- ***Interpreted code*** (the RCX byte code, pbFORTH)
  - requires an interpreter at the run-time system (increases space requirements)
  - allows higher level coding – for example, “motor off” instead off “increment X (decreases space requirements)
  - has interpretation overhead (decreases speed)
  - may have restricted functionality relative to native code
  - is portable
- ***Native code*** (running legOS)
  - no restriction on hw capabilities
  - no interpretation overhead
  - not portable

A Warning

- As is typical for embedded systems,
  - there are no built-in protection mechanisms
  - there is no “protected mode”
- Firmware *and any application* may read or write any address
- ***The good news:***
  - If all fails, we can take out the batteries – and start over again;
  - There is no way to destroy the boot loader stored in ROM

And there is more ...

- Several ***Java*** virtual machines
- RCX has also been programmed in
  - ***Ada*** (via an Ada2NQC translator)
  - ***Scheme***
  - ***C++***
  - ***Basic***
- Users have also written
  - a utility to program the RCX using just the RCX buttons
  - a Jini/Mindstorms driver

To Go Further

- ***Mindstorms:***
  - ☞ Links from class homepage
  - ☞ Jonathan Knudsen, “Lego Mindstorms”, O'Reilly, 1999

Problem Set 6 – Due: 30 May 2002

Build a Mindstorms robot that performs the following tasks:

- Upon program start, the robot starts moving forward.
- As soon as the robot (completely) crosses a dark line, it stops.
- The robot measures and displays the following values:
  - *T*: the time from start to stop;
  - *D*: the distance from start to stop; and
  - *L*: the thickness of the crossed line.

a) *Documentation* (overview of approach and assumptions, commented source code) **(3 pts)**

b) Functional *robot* **(2 pts)**

c) Why is this a *real-time problem*? **(1 pt)**

d) What are the real-time *entities/images/representatives* involved? **(2 pts)**

e) What can you say (qualitatively and quantitatively) about the *errors* in your measurements? **(3 pts)**

Please bring your robot along for the homework discussion on Tuesday.

Enjoy!