Introduction to Lab 2
Programming LEGO Mindstorms NXT using Ada

Jakaria Abdullah

19 September 2017
Lab 2: Programming LEGO NXTs using Ada

- Lab goals:
  - Real-time programming on an embedded device
  - Using concurrent Ada tasks to solve a problem

- Lab organization:
  - Work in your groups
  - Get LEGO box after this lecture, charge battery
  - Labs will be done on 2315 (building 2 floor 3)
    - Thu, 21.9, 08:15-12:00
    - Thu, 28.9, 13:15-17:00
  - Participate in demonstration on 02.10. in 2315
  - Have a look at the lab homepage
    http://www.it.uu.se/edu/course/homepage/realtid/ht17/lab2

- Lab report:
  - Ada code to all parts, well commented
  - Submission page in studentportalen; Deadline: Mon, 02.10. at 23:59
Lab 2: LEGO NXT Boxes

- Each group gets one box
- All group members are responsible for box parts
- For missing or faulty hardware issues, report to lab assistants
- Return all hardware to assistants on **Mon, 02.10**
- Check instructions in the box, to arrange items
- Lab assistant will flash the default firmware
You may work at home (using Windows/Linux/Mac?)

Toolchain installation is non-trivial
- Firmware upload, program compile, program upload
- Windows: Need Cygwin
  - *I can’t give support for that*
- An instruction file is available at lab homepage

External toolchain sources:
- *Linux toolchain from UPM*
- *Telecom Paristech Lego corner*

Default: Work in the Windows lab (2315)
LEGO Mindstorms

- Programmable LEGO brick with sensors and motors
- Comes in several generations:
  - RCX generation (1998)
  - NXT generation (2006)

- We will use the *NXT platform*
LEGO Mindstorms: Components

Package contents:

- **NXT unit:**
  - LCD matrix display
  - Sensor inputs 1 to 4
  - Motor outputs A, B, C
  - Speaker
  - USB, Bluetooth

- Three motors

- Sensors:
  - Light
  - Distance (Ultrasound)
  - Touch (2x)
  - Sound
  - (More from 3rd party vendors)

NXT Brick Internals:

- 32-bit ARM7 main processor, 8-bit AVR co-processor, 64k RAM, 256k Flash, 48MHz clock
We don’t use the standard firmware

Instead: *Ada runtime system for NXT*

- Not a real-time operating system
- Supports
  - Concurrent tasks, priorities,
  - Fixed priority preemptive scheduling
  - Resource sharing using protected objects
  - Drivers for low-level I/O accesses
- Only 4186 lines!
- Program, drivers and runtime system can run together inside the RAM!

Based on Ravenscar Small Footprint Profile (SFP)

Ravenscar is restricted subset of Ada suitable for static analysis
Ada NXT Runtime: Quick Look

- **GNARL**: Ada runtime library which imposes Ravenscar restrictions
- **GNULL**: Low-level library that needs to be ported for different architecture
- **Drivers**: For communicating with the sensor and actuators
Ravenscar Profile

- **Background:**
  - Conceived at IRTW 1997 at Ravenscar, Scotland
  - Main idea is to restrict Ada features for predictability

- **Some restrictions:**
  - Task type and task object declarations *only in library level* packages
  - At most *one entry* for each protected object/type
  - Each entry barrier expression must be *single Boolean* variable
  - At most one task at a time may be queued on an entry
  - *No select* statements and no task entries
  - No relative delay (delay) statements (use delay until)
  - No references to package Ada.Calendar (use Ada.RealTime)
  - No 'Image and 'Value attribute
Program Compile

1. Use cygwin terminal to compile programs in Windows
2. May need to *adjust* the provided Makefile
3. Compile program using `make all`
4. Clean up compilation by `make clean`

Example Run: Program compile via Cygwin terminal

```
$ cd /cygdrive/c/gnat/2012/test-folder
$ make all
...
A lot of compiler messages will be generated, a successful compilation will have no error
...
$ make clean  # optional, but useful
...
$
```
Program Upload

1. Connect NXT unit to USB port
2. Power up NXT unit
3. Put NXT into *reset mode*
4. Ticking sound means NXT is ready to upload
5. Upload program using *samba*

Example Run: Program upload via Cygwin

```bash
$ cd /cygdrive/c/gnat/2012/test-folder
$ samba_run event_driven
...
Image download complete.
Image started at 0x0020235c
......
$
```
Program Upload

1. Connect NXT unit to USB port
2. Power up NXT unit
3. Put NXT into reset mode
4. Ticking sound means NXT is ready to upload
5. Upload program using samba

Example Run: Program upload via Cygwin

```bash
$ cd /cygdrive/c/gnat/2012/test-folder
$ samba_run event_driven
...
Image download complete.
Image started at 0x0020235c
......
$
```
Program Structure

You “program” three files:

1. A main procedure file: *ads source file*
2. Task specification package file: *ads source file*
3. Task implementation file: *adb source file*

Main Procedure File:
- Staring point of your program
- This procedure name will be the name of your compiled program

Task specifications:
- Define global variables, task periods, etc
- Initialize sensors and motors with proper ports

Task implementations:
- Task body code for implementation

Will do a short walk-through now

See the lab web page for example codes
Program Structure

- You “program” three files:
  1. A main procedure file: *ads source file*
  2. Task specification package file: *ads source file*
  3. Task implementation file: *adb source file*

- Main Procedure File:
  - Staring point of your program
  - This procedure name will be the name of your compiled program

- Task specifications:
  - Define global variables, task periods, etc
  - Initialize sensors and motors with proper ports

- Task implementations:
  - Task body code for implementation

- Will do a short walk-through now

- See the lab web page for example codes
Program Structure

- You “program” three files:
  1. A main procedure file: *ads source file*
  2. Task specification package file: *ads source file*
  3. Task implementation file: *adb source file*

- **Main Procedure File:**
  - Staring point of your program
  - This procedure name will be the name of your compiled program

- **Task specifications:**
  - Define global variables, task periods, etc
  - Initialize sensors and motors with proper ports

- **Task implementations:**
  - Task body code for implementation

- Will do a short walk-through now

See the lab web page for example codes
Program Structure

- You “program” three files:
  1. A main procedure file: *ads source file*
  2. Task specification package file: *ads source file*
  3. Task implementation file: *adb source file*

- Main Procedure File:
  - Staring point of your program
  - This procedure name will be the name of your compiled program

- Task specifications:
  - Define global variables, task periods, etc
  - Initialize sensors and motors with proper ports

- Task implementations:
  - Task body code for implementation

- Will do a short walk-through now

See the lab web page for example codes
I/O Interfacing

- Input via orange button and sensors
  - Initialize sensors before use
- Output via LCD (strings, integers), sound and motors
- Sensor and motor access via ports: Motor_A, ..., Sensor_1, ...
- See examples!

Example: Initializing sensors and motors

```
1 Touch_Sensor_Id : Touch_Sensor (Sensor_1);
2 Light_Sensor1 : Light_Sensor := Make(Sensor_2, True);
3
4 Right_Motor_Id : constant Motor_Id := Motor_A;
5 Left_Motor_Id : constant Motor_Id := Motor_B;
```
Motor control

- Motor_control API provides high level functions for motor movements

Example: Using motor control API

```plaintext
procedure Forwards is
begin
  Control_Motor (Right_Motor_Id, Speed_Full, Forward);
  Control_Motor (Left_Motor_Id, Speed_Full, Forward);
end Forwards;

procedure Turn_Left is
begin
  Control_Motor (Right_Motor_Id, Speed_Full, Backward);
  Control_Motor (Left_Motor_Id, Speed_Half, Backward);
end Turn_Left;
```
Motor control

- More precise control of motor operation can be done by using nxt-motors-simple and nxt-motors-encoder drivers

Example: Testing motor encoders

```plaintext
procedure Test_Motor_Encoder is use NXT;

Engine : Simple_Motor := Make (Motor_A);

begin
    Engine.Set_Power (50);
    Engine.Forward;

    loop
        Put_NoUpdate (Encoder_Count (Motor_A)); New_Line;
        exit when Current_Button /= No_Button;
        delay until clock + milliseconds (100);
    end loop;

    ....
```
Display Functions

- nxt-display driver provides API for LCD display

Example: API for LCD display

```plaintext
procedure Clear_Screen_Noupdate;
procedure Clear_Screen;
procedure Set_Pos (Column : Char_Columns; Row : Char_Rows);
  -- Set current position.

procedure Put_Noupdate (C : Character);
procedure Put_Noupdate (S : String);
procedure Put_Noupdate (V : Integer);
procedure Put_Noupdate (V : Long_Long_Integer);
  -- Write a character, a string and an integer.
  -- Only CR and LF control characters are handled.
  -- Note that the min and max values for
    Long_Long_Integer will wrap around the display.
```
Display Functions

Example: More API for LCD display

1. `procedure Put (C : Character);`
2. `procedure Put (S : String);`
3. `procedure Put_Line (S : String);`
   -- Like in Ada.Text_IO.

4. `procedure Newline_Noupdate;`
5. `procedure Newline;`
6. `procedure New_Line renames Newline;`
7. `procedure New_Line_Noupdate renames Newline_Noupdate;`
   -- Like in Ada.Text_IO.

8. `procedure Screen_Update;`
   -- Synchronize the LCD with the internal buffer.
Synchronization Features

- Synchronization between tasks by protected objects
- Ravenscar restriction: *only one entry* per protected object
- Protected object itself should have priority at least as high as the maximum priority of the user tasks
- Access to this protected object is controlled by Immediate Ceiling Locking Protocol (ICPP)
Example: Ravenscar-compliant Protected Object

```vhdl
protected Event is

  entry Wait(D : out Integer);
  procedure Signal(D : in Integer);

private

  -- protected object priority, ceiling of the user priorities
  pragma Priority(System.Priority’First + 3);

  -- protected object data declaration
  Current : Integer;

  -- barrier variable for the entry
  Signalled : Boolean := False;

end Event;
```
Example: Ravenscar-compliant Protected Object

```plaintext
protected body Event is
  entry Wait (D : out Integer) when Signalled is
  begin
    D := Current;
    Signalled := False;
  end Wait;

  procedure Signal (D : in Integer) is
  begin
    Current := D;
    Signalled := True;
  end Signal;
end Event;
```
Lab Assignment

- **Part 1: Warm-Up**
  - Attach only light sensor
  - Write light values
  - Nothing fancy, just to get a soft start

- **Part 2: Event-driven Scheduling**
  - Communication between tasks using protected object
  - Application: Four events with car on table
    - Touch sensor is pressed/released
    - Table edge is sensed (light sensor)

- **Part 3: Periodic Scheduling**
  - Define different periodic tasks
  - Application: Sensor readings with periodic tasks
    - Drive backward if table edge is detected
    - Drive forward when button is pressed and on table

- **Last Part: Line Tracker**
  - Apply all you have learned
  - (See next slide)
Lab Assignment

- **Part 1: Warm-Up**
  - Attach only light sensor
  - Write light values
  - Nothing fancy, just to get a soft start

- **Part 2: Event-driven Scheduling**
  - Communication between tasks using protected object
  - Application: Four events with car on table
    1. Touch sensor is pressed/released
    2. Table edge is sensed (light sensor)

- **Part 3: Periodic Scheduling**
  - Define different periodic tasks
  - Application: Sensor readings with periodic tasks
    1. Drive backward if table edge is detected
    2. Drive forward when button is pressed and on table

- **Last Part: Line Tracker**
  - Apply all you have learned
  - (See next slide)
Lab Assignment

- **Part 1:** *Warm-Up*
  - Attach only light sensor
  - Write light values
  - Nothing fancy, just to get a soft start

- **Part 2:** *Event-driven Scheduling*
  - Communication between tasks using protected object
  - Application: Four events with car on table
    1. Touch sensor is pressed/released
    2. Table edge is sensed (light sensor)

- **Part 3:** *Periodic Scheduling*
  - Define different periodic tasks
  - Application: Sensor readings with periodic tasks
    1. Drive backward if table edge is detected
    2. Drive forward when button is pressed and on table

- **Last Part:** *Line Tracker*
  - Apply all you have learned
  - (See next slide)
Lab Assignment

- **Part 1: Warm-Up**
  - Attach only light sensor
  - Write light values
  - Nothing fancy, just to get a soft start

- **Part 2: Event-driven Scheduling**
  - Communication between tasks using protected object
  - Application: Four events with car on table
    - 1. Touch sensor is pressed/released
    - 2. Table edge is sensed (light sensor)

- **Part 3: Periodic Scheduling**
  - Define different periodic tasks
  - Application: Sensor readings with periodic tasks
    - 1. Drive backward if table edge is detected
    - 2. Drive forward when button is pressed and on table

- **Last Part: Line Tracker**
  - Apply all you have learned
  - (See next slide)
LEGO Line Tracker

- *Car demonstration* takes place on Mon, 02.10.
- Track looks roughly like this:

![Track Diagram]

- Objectives:
  1. Follow the line and complete a lap
  2. Try to be fastest, finish the lap within maximum 1 minute
- 3 tries per team (otherwise: assignment failed, fix car)
- Keep in mind: Demo conditions might differ (different light etc.)
Some Additional Pointers

- Try to compile demo programs available in:
  /gnat/2012/share/examples/mindstorm-nxt/demos

- Details of available drivers are in:
  /gnat/2012/lib/mindstorm-nxt/drivers

- More information about NXT motors:
  http://www.philohome.com/nxtmotor/nxtmotor.htm

- Useful tutorials about line follower Lego Robot:
  http://www.nxtprograms.com/line_follower/steps.html
  http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html
The End

Questions?