Introduction to Lab 2
Programming LEGO Mindstorms NXT using Ada

Jakaria Abdullah

12 September 2016
Lab 2: Programming LEGO Mindstorms using Ada

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

- **Lab preparation:**
  - Work in your groups
  - *Get LEGO box* (next slide), charge battery
  - Possibly refresh your Ada knowledge
  - Lab will be done on Thu, 15.9. and Mon, 19.9. (both in 2315)
  - Have a look at the lab homepage
    http://www.it.uu.se/edu/course/homepage/realtid/ht16/lab2

- **Lab report:**
  - Ada code to all 3 parts, well commented
  - Descriptions of what you did and why
  - To submission page in studentportalen; *Deadline: Mon, 26.9. at 12:59*

- **Further:**
  - Demonstrate a working vehicle, participate in *demonstration on 26.9. in 2315*
  - Return all hardware you get to Karl (see next slide)
Lab 2: LEGO Mindstorms Boxes

- Each group gets one box
- All hardware issues are handled by Karl Marklund
- Office: 1440, mail: karl.marklund@it.uu.se
Lab 2: Working At Home

- You may work at home (using Windows/Linux/Mac?)
- Toolchain installation is non-trivial
  - *I can’t give support for that*
  - Firmware upload, program compile, program upload
  - Windows: Need Cygwin
- An instruction file is available at lab homepage
- *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
Programming Environment: Ada NXT

- We don’t use the standard firmware
- Instead: *Ada runtime system for NXT*
  - Not a real-time operating system
  - Based on Ravenscar Small footprint profile (restricted subset of Ada suitable for static analysis)
  - Can run without OS, only 4186 lines!
  - Support for (concurrent) tasks, priorities, protected object and fixed priority preemptive scheduling
  - Drivers for low-level I/O accesses
  - Program, drivers and runtime system can run together inside the RAM!
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

- **Restrictions:**
  - No locally declared task objects or task types and only locally declared protected objects or protected types
  - No dynamic allocation of task objects or protected objects
  - At most one entry for each protected object/type
  - Each entry barrier expression must be a single Boolean variable
  - At most one task at a time may be queued on an entry
  - No task termination and no use of dynamic priorities
  - No requeue and abort statements or Asynchronous Transfer of Control
  - No select statements and no task entries (thus no accept statements)
  - No relative delay (delay) statements
  - No references to package Ada.Calendar
  - No user defined task attributes
  - No use of dynamic priorities
  - No 'Image and 'Value attribute
Allowed Ada Features:

- Task type and task object declarations in library level packages
- Protected type and protected object declarations in library level packages
- Absolute delay (delay until) statements
- References to package Ada.RealTime
- Protected procedures as interrupt handlers
- FIFO Within Priority dispatching policy
- Ceiling Locking locking policy
- Pragmas Atomic and Volatile for shared data
- Count attribute (but not in a barrier expression)
- Discriminants for protected types and task types
Use cygwin terminal to compile programs in Windows
May need to *adjust* the provided Makefile
Compile program using `make all`
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

```
$ 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:
  - Starting 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

```plaintext
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
6 Left_Motor_Id : constant Motor_Id := Motor_B;
```
Motor control

- Motor control API provide 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 provide API for LCD display

Example: API for LCD display

```pascal
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.
```
Example: More API for LCD display

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

procedure Newline_Noupdate;
procedure Newline;
procedure New_Line renames Newline;
procedure New_Line_Noupdate renames Newline_Noupdate;
   -- Like in Ada.Text_IO.

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

- Synchronization between tasks can be done 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).
Synchronization Features

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;
```

Jakaria Abdullah
Lab 2: LEGO
12 September 2016
Synchronization Features

Example: Ravenscar-compliant Protected object

```vhdl
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;
```
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: Distance and touch sensor sensing
  1. Drive (back off) while sensor pressed
  2. Otherwise, keep distance constant

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: Distance and touch sensor sensing
    1. Drive (back off) while sensor pressed
    2. Otherwise, keep distance constant

- **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: Distance and touch sensor sensing
    1. Drive (back off) while sensor pressed
    2. Otherwise, keep distance constant

- **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: Distance and touch sensor sensing
    1. Drive (back off) while sensor pressed
    2. Otherwise, keep distance constant

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

- *Car demonstration* takes place on Mon, 26.9.
- Track looks roughly like this:

![Track Diagram](image)

- Procedure for each team:
  1. **1st phase:** Follow another car in constant distance (20cm) for 1 lap
  2. **2nd phase:** Be fastest on the next lap
- Fastest team wins! (*Prize award included*)
- 3 tries per team (otherwise: assignment failed, fix car)
- Keep in mind: Demo conditions might differ (different light etc.)
Some Additional Pointers

- 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?