Synchronous programming exercises
(programming)

Pascal Raymond
Verimag-CNRS

MOSIG - Embedded Systems

Environment

The tools are available on the EnsiPc machines running CentOs (e.g. room E100).

To setup your environment, copy these lines on your .bashrc file:

```
export LUSTRE_INSTALL=/user/5/raymond/lustre
source $LUSTRE_INSTALL/setenv.sh
```

Environment
First steps with Lustre

Rising edge node

Write a node detecting the rising edges of its Boolean input. The profile should be:

\[ \text{node edge}(x: \text{bool}) \text{ returns } (e: \text{bool}); \]

Simulation

Use the graphical simulator for testing the program:

```
  luciole tp.lus edge
```

Or simply `luciole` then browse.

- try the "clock" options (auto-step vs compose, real-time clock)
- try the associated "tools" (sim2chro)

Write and simulate other nodes

For instance the ones presented in the course (switch, counter, stopwatch etc).

---

Car lights controller

behaviour

- from "all lights off", turn left (TL) sets side lights,
- from "side lights", turn left switches off the side lights and sets low lights,
- from low or high lights, pulling the lever (LH) switches between low and high,
- turning right in low/high state returns to side lights,
- turning right in side state returns to "all lights off".
Controller

Write, test, simulate the controller:

To go further:

- add a "fog lamp" functionality, controlled by a check button, and effective only in low lights mode
- add a "long range lamp" functionality, controlled by a check button, and effective only in high lights mode

---

The reverse pendulum

A typical example involving numerical computing and "signal processing method".

Principle

![Diagram of the reverse pendulum]

- Forces:
  - $m\vec{g}$ (weight), $\vec{f}$ (reaction)

- Geometry:
  - $x = x_0 + l \sin(\theta)$,
  - $y = y_0 + l \cos(\theta)$

Newton’s equations: $m\vec{g} + \vec{f} = m\ddot{\gamma}$
Mathematical model

• Tangential and radial acceleration \( \gamma = \gamma_r + \gamma_t \)
  with: \( \gamma_t = x'' \cos(\theta) - y'' \sin(\theta) \)
• Projection on tangent: \( \gamma_t = g \sin(\theta) \)
• And (basic geometry):

\[
\begin{align*}
x' &= x'_0 + l \sin(\theta) \theta' \\
x'' &= x''_0 - l \sin(\theta) \theta'^2 + l \cos(\theta) \theta'' \\
y' &= y'_0 - l \sin(\theta) \theta' \\
y'' &= y''_0 - l \cos(\theta) \theta'^2 - l \sin(\theta) \theta''
\end{align*}
\]

The reverse pendulum

Mathematical model (contd)

• Substitution ... \( g \sin(\theta) = x''_0 \cos(\theta) - y''_0 \sin(\theta) + l \theta'' \)
• And finally:

\[ \theta'' = ((y''_0 + g/l) \sin(\theta) - (x''_0/l) \cos(\theta) \]

The reverse pendulum
Programming a numerical library

For a given (constant) sampling period of $T$ seconds, write:

- a discrete derivative node: `node D(x:real) returns (dx:real)`
  hints: the discrete derivative if the slope

- a discrete integrator node: `node I(dx:real) returns (x:real)`
  hints: the integral is the surface area between the curve and the axis, it can be approximated by accumulation small rectangles (or trapezes) areas.

- a delayed discrete integrator node: `node ID(dx:real) returns (x:real)`
  such that $x$ does not depend instantaneously on $dx$?

Programming the pendulum equation

Program `directly` the equation with a node that:

- takes as input the acceleration of the basis point $d2x0, d2y0$
- computes the current angle $\theta$

  `node pend( d2x0,d2y0:real) returns (teta:real);`

Programming a game based on the pendulum

The player tries to stand in balance a stick on the palm of his hand:

- the inputs are the coordinates of the basis of the stick $(x0, y0)$,
- the outputs are the coordinates of the top of the stick $(x, y)$
  `node game(x0,y0:real) returns (x,y: real)`
Running the program ...

- Using luciole is not convenient for this example.
- We provide an ad-hoc main graphical program written in tcl/tk.
- Download the necessary files here:
  http://www-verimag.imag.fr/~raymond/edu/mosig/pendulum.tgz

Warning!

- The program file must be called game.ec,
- Use lus2ec my_program.lus game to create it (or see the given Makefile),
- the sampling period in the lustre program (e.g. 0.02 s) must be coherent with the one of the tcl/tk program (given in ms, e.g. 20)
- The length of the pendulum should be 4.0.

Remarks

- the shorter is the period, the smoother is the simulation,
- ... but the execution method used here (interpreter + unix pipes) is rather inefficient, and don’t support high rates (50 Hz, i.e. 20 ms is reasonable).
Adding a frictional damping force

The simulation is quite unrealistic, cause the pendulum cannot lose kinetic energy.

- Think about a way for introducing some frictional damping force in the equation.
- hints: a simple approximation consist in introducing a damping force proportionnal to the angular velocity, the Newton’s Equation becomes:

\[ g \cdot \sin(\theta) - x_0'' \cdot \cos(\theta) + y_0'' \cdot \sin(\theta) - l \cdot \theta'' - a \cdot \theta' = 0 \]

- Try with different values of \( a \).

Programming with Esterel

Mouse click detector

- two "clicks" separated with less than 5 basic-clock ticks are considered has a "double click", otherwise it is a simple click.
- copy the code in a file `mouse.strl`

```plaintext
module mouse:
  input click;
  output single, double;
  loop
    await click;
    abort
      await 5 tick; emit single
    when click
      do emit double end
  end.
```

Programming with Esterel 13/15
Running the Esterel program

An Esterel program can be simulated using luciole:

- call the script `esterel2dro mouse.strl mouse`
  builds a dynamic library `mouse.dro`, in a format recognizable by luciole
- run `luciole` and load `mouse.dro` to start the simulation.

Visualizing the Esterel program semantics

The automaton of an Esterel program can be explored using atg:

- call the script `esterel2 mouse.strl mouse`
  compiles the program into an automaton `mouse.atg`
- run `atg mouse.atg`, then press 'x' to start the exploration.

To go further

- Write and simulate the examples seen in the course.
- Write an Esterel version of the car lights controller
  
  hints:
  
  - the `sustain X` statement is a (convenient) shortcut for `loop emit X; pause end`
  
  more generally, see the slides on Esterel to find a list of Esterel statements.