High-level Formal programming

MAE-Embedded Systems

Robot Lego Mindstorms EV3

Frederic BONIOL frederic.boniol@onera.fr
Christine Tasson Christine.TASSON@isae-supaero.fr
Xavier THIRIOUX Xavier.THIRIOUX@isae-supaero.fr

The gooal of this BE is to program in Lustre the controler of a robots avoiding obstacles and stopping when arriving in dark zones.

Milestone and Markings

This BE will be marked, you have to submit before **Friday, Feb 10, 6pm** on the LMS an arxiv containing: the Lustre files with, if needed `.c` and `.h` files and a file `Readme.md` explaining how to test the different nodes. Your code have to be documented: the constants, the interface of the nodes and the conception have to be explained as comments.

We give you a proposition of Software development Milestone with Markings indications:

M1 (4 points) The robot goes forward for a while and changes direction: backward, right and left, and starts again.

M2 (4 points) Add the color sensor: the Robot stops when it crosses a dark zone.

M3 (4 points) Add Left and Right touch sensors.

M4 (4 points) Add Ultrasonic Sensor.

M5 (4 points) Use all sensors together with clocks.

For each Milestone, the evaluation will take into account both if the task is achieved and if the code is documented with an HowTo (use Readme.md) and with comments in the code explaining constants, inputs, outputs, variables and intermediate nodes.

1 Description of brick of the Lego Mindstorms EV3

Figure 1: Lego Mindstorm EV3 brick

The robot that you are about to program is using the brick Lego Mindstorm EV3, released in 2013. It contains:

- A monochromatic screen LCD 178x128px.
- A processor Texas Instrument AM1808 mono-core, relying on ARM9 with a frequency of 300 MHz.
- A main memory of 64 MB, a Flash memory of 16 MB, and a place to put an SD card of 32 MB max.
- Two USB ports, a WIFI connexion, a Bluetooth connexion.
- Four ports, numbered from 1 to 4, to connect Lego sensor.
- Four ports, numbered from A to D, to connect Lego motors.

A Linux kernel has been installed on the SD card together with open-source libraries to read sensors and actuate motors. The libraries, the drivers and several examples of programs are available at address https://github.com/in4lio/ev3dev-c. Tutorials and documentation for this brick are available at address https://www.ev3dev.org.

2 Description of the robot

The goal of the work is to program the robot's processor to drive the wheel motors and the ultrasonic sensor's pivot motor, according to the information received from the touch sensors (right and left), the ultrasonic sensor (front, right and left) and the light sensor (behind).

2.1 Material architecture: sensors and actuators

The robot is presented in Fig. **??**. It is made of:

- Two driving wheels: a right wheel and a left wheel (the rear balance is ensured by a non-driving ball).
- Two touch sensors, which by means of a "bumper" type device (or whisker), detect a collision with an obstacle. These two sensors are respectively in front of the bottom right and in front of the bottom left. When a sensor is pressed it sends a "true" signal to the port where it is plugged in. When the sensor is not pressed, it sends a "false" signal.
- An ultrasonic sensor located in height allowing to measure the distance in mm to the next obstacle. This sensor is mounted on a mobile device that can be turned to the right and left. By turning this device, it is possible to measure the free distance:
  - in front of the robot,
  - to the right of the robot,
  - and to the left of the robot.

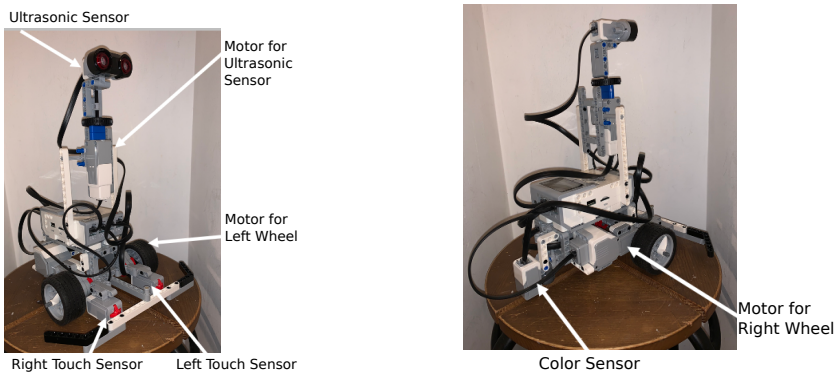


Figure 2: Front and back view of the Robot

- A color sensor located behind at ground level. This sensor returns a value between 0 and 100: 0=black, and 100 = very bright white. Very dark spots detected by this sensor generally return a value between 5 and 10.
- Two motors to turn the right and left wheels. The wheel motors are controlled in speed, forward or backward.
- A motor to turn the ultrasonic sensor. The motor linked to the ultrasonic sensor is controlled in position. The 0 position corresponds to the initial position when the program is launched (remember to position the sensor by hand so that it "looks" forward so that this position corresponds to the zero position). From this 0 position, it is possible to rotate the motor by X degrees to the right or X degrees to the left. This value of X should not be greater than 180 degrees (because of the cable connecting the ultrasound sensor to the Lego brick).

The sensors and motors are connected as follows:

- The right wheel motor is connected to port A.
- The left wheel motor is connected to port B.
- The motor that rotates the ultrasonic sensor is connected to port C.
- The right touch sensor is connected to port 1.
- The left touch sensor is connected to port 4.
- The ultrasonic sensor is connected to port 3.
- The color sensor is connected to port 2.

2.2 Software architecture

The Lego EV3 brick integrates an ARM9 processor under Linux. The control program will be executed on this processor. This program is named `robot_isae.c`. It is written in `C`. Its architecture is described below:

1. Initialization of the robot, test of the presence of the motors and sensors, identification of the ports on which they are connected. Initialization of the boolean " alive " to true.
2. While alive=true
   (a) Reading of the values on the sensors (touch, ultrasound, color)

(b) Execution of the control function (calculation of the commands to be sent to the three motors according to the value of the sensors and the current state of the robot).

(c) Sending the commands to the three motors.

(d) Test of pressing the "back" button: if pressed then alive=false, otherwise nothing.

(e) Wait for 10ms.

3. End While

This program is organized as a pseudo-periodic loop of at least 10ms. The intelligence of the program is in line 2.(b). It is this part that you have to realize in Lustre. All the other parts of the program are written in `C` (sequential), and are already provided. You will only have to write the Lustre code for the line 2.b. (But if you want to see and modify the C code, the `robot_isae.c` program is accessible and open).

2.3 Mission to achieve

The goal of this robot is to navigate randomly in a room until it finds a black (or very dark) area on the floor and stops there.

To do this, it drives straight ahead:

- Until it encounters an obstacle, in which case he tries to avoid it.
- Or until it detects a black (or very dark) area, in which case it stops,
- Or until a timer is triggered, telling him that he has already driven a lot forward without encountering an obstacle, in which case he decides to turn right (or left, it's up to you) and then starts again by reactivating the timer. The obstacle detection is done via the touch sensors and the ultasonics sensor.
- If the right touch sensor has been pressed (an obstacle has been encountered on the right), then
  - the robot switches off its wheel motors, and waits until the wheels are actually stopped,
  - backs up for a certain time (duration to be chosen),
  - switches off the motors of its wheels again, and waits again for the wheels to be effectively stopped,
  - turns to the left (since the obstacle is on the right) for a certain time (duration to be chosen)
  - turns off the motors of its wheels again, and waits again until the wheels are actually stopped,
  - and goes straight ahead again.
- If the left sensor was hit, but turning to the right, do the same.
- If the ultrasonic sensor has detected the presence of an obstacle in front of it within 20 centimeters, then
  - the robot turns off its wheel motors, and waits until the wheels are actually stopped,
  - turns its ultrasonic sensor to the right to measure the free distance to the right,
  - same thing to the left,
  - and decides to go in the direction (right or left) with the most clearance.

Note: to turn right, simply command the right wheel backward and the left wheel forward. The same goes for turning left (right wheel forward and left wheel backward). To stop the motor, the two booleans `CLOCKWISE` and `COUNTERCLOCKWISE` must both be at zero.

One of the difficulties is to respect the inertia of the motors: before changing a steering motor, it is necessary to wait until it is stopped. Each motor contains a speed sensor that can be measured. It will always be necessary to wait for this sensor to return to zero before changing the direction of rotation of the motor.

2.4 Lustre program architecture

You will find on the LMS a skeleton of the Lustre program `main_robot.lus`.

Remark: it is imperative that the main node of your program is named `main_robot` (otherwise, you will have to change the `C` program `robot_isae`).

It is also imperative to respect the naming of the input and output streams above. These streams are written and read by the `robot_isae` program.

Remark : the robot we want to program contains only three motors. The output D is therefore not used. But for the sake of scalability, your program will still return streams for this motor D (by setting them to `false` (for boolean streams) and `zero` (for integer streams)).

3 Work to do

Your job is to design and code in Lustre the `main_robot` node that codes the control function of the robot.

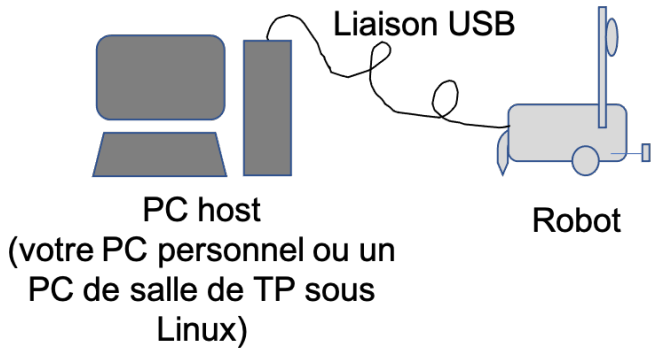
Tip: To program this robot, you will probably need nodes seen in Session 2:

- a timer that counts down the time like `stable`
- a rising or falling edge detector
- an RS switch

4 Compilation, execution

The development cycle that you are going to follow has two stages:

- On the one hand, the Lustre part itself, realized on the "host" PC (your workstation), and during which you will use the tools of the Lustre toolbox.
- And on the other hand, the copy of the generated code (the C code) on the robot, its compilation and its execution.



Step 1: On the host PC

Step 1 consists in designing and developing the `main_robot` node (and the sub-nodes called by it if you think it is necessary).

Instructions:

- call your Lustre file `lego_robot.lus` (provided on LMS)
- indicate at the beginning of the file in comment your first and last names

When you want to test your program on the robot, it is necessary to generate the corresponding `C` code. To do this, you will use `lus2c` to generate `main_robot.c` and `main_robot.h`

```
lus2c lego_robot.lus main_robot -o main_robot.c
```

If your Lustre file contains external objects (constants not initialized in the lustre file, or external nodes), you have to define them in an additional file, written in C, called :

- `main_robot_ext.h`

If your Lustre code contains clocks, you first need to transform it into Lustre v4, before compilation:

```
lv6 lego_robot_clock.lus -node main_robot -lv4 -o lego_robot.lus
lus2c lego_robot.lus main_robot -o main_robot.c
```

Output of step 1:

- the file `lego_robot.lus` completed with your Lustre code.
- the generated code `main_robot.c` and `main_robot.h`
- the additional file `main_robot_ext.h` (if needed)

Step 2 : on the robot

The second step takes place on the robot's processor. We will connect to it via a remote `ssh` session.

To boot the robot, press the central button of the EV3 brick (this may take 1 to 2 minutes).

Connecting the robot to the host PC The robot connects to the host PC via a USB port. Connect the robot to a USB port on the host PC.

On the host PC, in the network settings, go to the USB Ethernet network connection. Go to the wired connection settings. Go to IPV4 and select `local network only`, disable DNS and Routes.

Then go to the robot interface. Locate the robot's IP address via the robot's screen by navigating with the buttons:

- Go to the `Wireless and networks` menu,
- then in `All network connections`,
- then `Wired`.
- If the status is Disconnected, click on `Connect`.
- When the robot is `Connected`, go to `IPV4` and note the IP address of the form `ip_address=ip_address` where `---` are between 1 to 3 digits.

To check that the robot is connected and accessible from the host PC, do

```
ping ip_address
```

If it doesn't work, disconnect (on the host PC or on the robot) and try again. It should work eventually.

Login on the robot (via an ssh connection) As a reminder, the OS running on this processor is a Linux (Debian). To connect to it, you must use the account :

- Login: `robot`
- Pwd: `maker`

In a terminal on the host PC, connect to the robot via ssh by doing

```
ssh robot@ip_address
```

It is possible that your PC asks you :

```
Are you sure you want to continue connecting (yes/no)?
```

You answer `yes`, then you enter the password (`maker`). You should see in your terminal the following result:

```
Linux ev3dev 4.14.117-ev3dev-2.3.5-ev3 #1 PREEMPT Sat Mar 7 12:54:39 CST 2020 armv5tejl
             _____     _
   _____   _|___ /  __| | _____   __
  / _ \ \ / / |_ \ / _` |/ _ \ \ / /
 |  __/\ V / ___) | (_| |  __/\ V /
  \___| \_/ |____/ \__,_|\___| \_/

  Debian stretch on LEGO MINDSTORMS EV3!
Last login: Fri Jan 30 15:43:27 2022 from fe80::7831:c1ff:fe8b:c164%usb0
robot@ev3dev:~
```

Your session is now running on the robot processor. Change directory by doing

```
cd /home/robot/ev3dev-c/eg/robot_isae
```

This will be your working directory (copy, compile and run) on the robot.
You will copy there the C code generated by Lustre.

Copy and compile the codes on the robot You have to copy and compile the codes (`.c` and `.h`) on the robot.
The copy of these codes is done from the host PC by :

```
scp main_robot.c robot@ip_address :ev3dev-c/eg/robot_isae/.
scp main_robot.h robot@ip_address :ev3dev-c/eg/robot_isae/.
scp main_robot_ext.h robot@ip_address :ev3dev-c/eg/robot_isae/.
```

There should already be another file in this directory called `robot_isae.c` This is the global `C` program, already mentioned earlier, which includes your Lustre program. You don't need to modify this file a priori. However, it must be present. If it is not, get it from `LMS` and copy it to the robot by the same procedure as the previous files.
To compile, you just have to use the makefile and do

```
make release
```

This command calls the `gcc` compiler for the `ARM9` processor of the robot on the different files. You may get warnings about unused variables in the `robot_isae.c` code. It is not your fault, so ignore these warnings.
This operation generates (if everything goes well) an executable in the Release directory.

Running the program To run the program, the easiest way is to do it in the terminal by going to the directory

```
cd /home/robot/ev3dev-c/eg/robot_isae/Release
```

Run the binary

```
./robot_isae
```

(be careful, your robot will probably start. Think of putting it on blocks so that its wheels do not touch the ground).

To stop the robot, press the `back` button on the EV3 brick.

The other way to launch the robot is via the robot interface:

- Go to `File Browser`.
- Go to `ev3dev-c`.
- Go to `eg`.
- Go to `robot_isae`
- Go to `Release`.
- And launch `./robot_isae`

Good work!