Lab 8: Introduction to the e-puck Robot

This laboratory requires the following equipment:

- C development tools (gcc, make, etc.)
- C30 programming tools for the e-puck robot
- The "development tree" which is a set of files required by the C30 compiler
- One e-puck robot, one Bluetooth USB dongle (if needed, a spare battery)
- The document "Introduction to the e-puck robot"

The laboratory duration is approximately 3 hours. Although this laboratory is not graded, we encourage you to take your own personal notes as the final exam might leverage results acquired during this laboratory session. For any questions, please contact us at **sis-ta@groupes.epfl.ch.**

1.1 Information

In this assignment, you will find several exercises and questions.

- The notation $\mathbf{Q_x}$ means that the question can be answered theoretically or with simple commands in the Linux operating system, without implementing or running any code.
- The notation $\mathbf{S_x}$ means that the question can be solved only by compiling and running a piece of code or an additional simulation.
- The notation $\mathbf{I_x}$ means that the problem has to be solved by implementing, possibly compiling, and running a piece of code.
- The notation $\mathbf{B_x}$ means that the question is optional (bonus) and should be answered if you have enough time at your disposal.

1.2 The e-puck

The **e-puck** is a miniature mobile robot developed to perform "desktop" experiments for educational purposes. Figure 1 shows a close-up of the **e-puck** robot. More information about the e-puck project is available at http://www.e-puck.org/.

Figure 1: Close-up of an **e-puck** robot.

The **e-puck's** most distinguishing characteristic is its small size (7 cm in diameter). Other basic features are: significant processing power (dsPIC 30F6014 from Microchip running at 30 MHz), programmable using the standard make compilation tools, energetic autonomy of about 3-4 hours of intensive use (with no additional modules),

an extension bus, a belt of eight light and proximity sensors, a 3-axis accelerometer, three microphones, a speaker, a color camera with a resolution of 640x480 pixels, 8 red LEDs placed around the robot and a Bluetooth interface to communicate with a host computer. The wheels are controlled by two miniature stepper motors, and can rotate in both directions. Figure 2 presents the location of the components, referred above, in the robot itself.

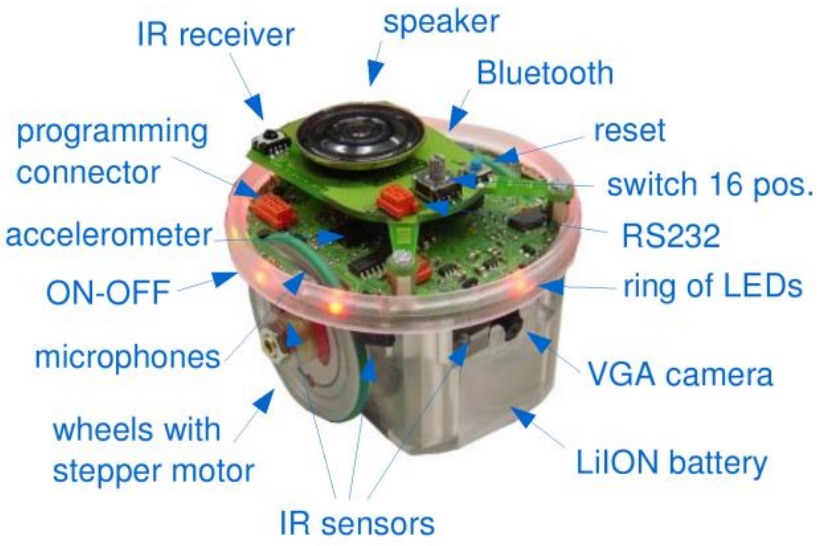

Figure 2: Review of the e-puck components.

The simple geometrical shape along with the positioning of the motors allows the **e-puck** to negotiate any kind of obstacle or corner.

Modularity is another characteristic of the **e-puck** robot. Each robot can be extended by adding a variety of modules. In this lab we further explore the capabilities of the **e-puck** hardware platform.

1.3 Getting Started (Short reminder)

To start with this lab, you will need to download the material available on Moodle. Download *lab08.tar.gz* in your personal directory. Now, extract the lab archive (you can type: **tar xvfz lab08.tar.gz**.)

1.4 Controlling the e-puck robot

In this lab, we will control the e-puck robot by running a program in the robot. This is done by cross-compiling a program for the e-puck on your computer. Cross-compilation is the process of compiling and building executable code for a platform other than the one on which the compiler is running. This can be useful for platforms upon which it is not feasible to do the compiling, such as microcontrollers that do not support an operating system. After creating the executable, you will then create a connection between your computer and the robot, and finally upload the program onto the robot's flash memory through the connection, using a wireless bootloader. The wireless bootloader code is already programmed on all the robots. The executable (a

.hex file) runs directly on the e-puck. The procedure to do this will be explained in detail in the following paragraphs.

2 Getting familiar with the e-puck

2.1 Preparing software and hardware

Before starting the lab, we need to set up the experimentation environment. Follow the instructions below to prepare hardware and software.

2.1.1 Preparing the software

You will need to download the e-puck development environment onto your computer. Copy these files in your home directory by doing the following:

a) First, go to a directory where you will place the e-puck related files. (example: `/home/user/Desktop/MyFiles`)
b) When in this directory, download (checkout) the necessary files from the e-puck git repository by typing the following (on one line):

```
> git clone
https://disalgitn.epfl.ch/epuck/epuck.git
```

You will be asked to accept the certificate permanently (option p). The download might take a few seconds.

c) Inside the `epuck/EpuckDevelopmentTree/` directory, you will find a `library/` and `program/` directory. In order to use the e-puck library code you just downloaded, it must be compiled. Do this by going to the library/ directory and executing *make*:

```
> cd EpuckDevelopmentTree/library
> make
```

Your e-puck development environment is now ready.

2.1.2 Preparing the hardware

Prepare your hardware setup now:
1. Plug in the Bluetooth USB dongle to your desktop computer.
2. Place the battery in the robot.
3. Switch the robot on.
4. Check your robot number written on a sticker at the front of the robot.
   My robot number is: ……

In the remainder of this lab, we assume that your robot ID is 999. **Make sure to replace 999 with the number of your robot.**

2.2 Basic sensor tests

We will start with some questions which should allow you to get a feeling of working with real hardware in general, and the e-puck in particular. To do this, you will

upload the *sercom* program onto the e-puck, and run the *minicom* program on your computer. *Minicom* is a small terminal program to communicate with a device over a serial link. Since the e-puck Bluetooth protocol emulates a serial link, we can use this program to chat with the e-puck.

First, create a connection to your robot:

```
> ssh localhost
> epuckconnect 999
```

Now navigate the directory lab08/ and upload *sercom.hex* onto the robot:

```
> epuckupload -f sercom.hex 999
```

In case you are prompted with a passkey dialog interface at the top of your screen: enter the **4-digit robot ID (e.g., 0999).**

As soon as dots appear on your screen (`$ .....`) press the blue reset button on the robot. This will start writing the program to the e-puck memory. Now stars will appear on your screen, indicating that the robot is being programmed (`.......** **********`). When no more stars appear, the robot has finished programming. You will see the following message in your terminal:

```
[/dev/rfcomm999] Transmission of sercom.hex complete!
```

Programming the e-puck might not work on the first attempt and you may have to try several times before succeeding. Now establish a communication channel with your e-puck and launch *minicom* by entering:

```
> epuckconnect 999 && minicom
```

Now press the reset button on the robot. Your terminal should display:

```
WELCOME to the SerCom protocol on e-Puck
the EPFL education robot type 'H' for help
```

With `Ctrl-A Z` (hold the `Ctrl` key while typing `A`, then release the `Ctrl` key and type `Z`) you get to the main menu of *minicom*. Configure your terminal to have a local echo (type `E`).

S_0: Enter `H <Enter>` in your minicom terminal. Your robot will answer with all the commands it understands through the serial line.

Most smartphones can detect their orientation by means of an accelerometer. The e-puck has such an accelerometer as well, allowing to measure acceleration (and therefore forces) in all three dimensions. For each dimension, you get a value between 0 and 4096. If no force is present along an axis, the corresponding value is roughly 2000 – 2300 (the exact center value depends on the e-puck).

S1: Put the e-puck on your desk and press `A <Enter>`. Then turn your e-puck on one side and measure the acceleration again. Why is one value out of the 2000 – 2300 range? Explain how a smartphone is able to detect its orientation.

Q2: What would you measure if the e-puck was in free fall (whatever experiment you do, please DON'T drop the e-puck on or from the table).

You can now quit *minicom* by typing `Ctrl-A X`.

2.3 Using the e-puck's IR sensors

In the last lab, you used the proximity and light sensors of a virtual e-puck to sense the proximity to nearby objects, or as a way to detect light. In this lab we test the same sensing capabilities in the real robot, leveraging its IR sensors.

2.3.1 Proximity sensors

In this part, we will use the IR sensors as proximity sensors. Recall the sensor distribution in the robot, described in the last lab.

S3: The code is in `e-puck_proximity` folder. The program gathers and transmits data from the proximity sensors. To compile the program, go to the directory and change the `EPUCKLIBROOT` variable in the Makefile to the `EpuckDevelopementTree/library` directory of your installation: (e.g., `EPUCKLIBROOT=~/Desktop/MyFiles/epuck/EpuckDevelopmentTree/library`)
Now you can compile your code by typing *make*. After that you can transfer the generated hex file (`e-puck_proximity.hex`) to the e-puck as done above.

Establish a connection and start a *minicom* terminal as before (`epuckconnect 999 && minicom`). Observe the readings. Why do they change over time? Can you relate to the previous lab?

S4: As you did in the previous lab, move an object around the e-puck (e.g., your hand), and draw the response of the proximity sensor as a function of the distance to an obstacle. How does it compare with the simulated sensor of the last lab?

S5: Now repeat question **S4** but with different objects (e.g., sheet of paper, pencil, pencil case, etc.) Explain why the values acquired for the same distances differ. Quit *minicom* by typing `Ctrl-A X`.

2.3.2 Light sensors

In this part we will use the IR sensors as light sensors. In the following exercise, you will use your e-puck to collect light intensity readings in different lighting conditions, and display them in MATLAB.

S6: The code is in `/e-puck_light` folder. The program gathers and transmits data from the IR receiver of the front right IR sensor (IR0). Compile and flash the program (`e-puck_light.hex`) to the e-puck, as explained above. Don't forget to add the path to the `EpuckDevelopmentTree/library` in the Makefile before compiling. Start collecting data by typing in the terminal:

```
> epuckconnect 999 && cat /dev/rfcomm999 | tee robot_data.txt
```

Cover and uncover this sensor with your hand and observe the changes in the value. Then point the sensor to an illuminated place (e.g., window) and then to a dark place (e.g., under the table). Again observe the changes in the value. Compare the variations happening in both situations and explain what you observe. Stop the data collection by typing into your terminal `Ctrl-C`.

S7: Similar to the previous lab, the light sensor readings are being written to a text file (robot_data.txt). Open Matlab, and plot the recorded values by typing:

```
load robot_data.txt
plot(robot_data);
```

Can you explain the graph based on the light sensor readings you saw previously in the robot window?

S8: You will now need to use the IR0 sensor to produce a signal to be analyzed. Start recording sensor data as above. Cover and uncover the sensor with your finger during 1 second intervals (cover during 1 second, uncover during 1 second, cover during 1 second, … , repeat this procedure multiple times). Plot the signal in Matlab. What type of signal is this? At which frequencies do you expect to see peaks if you plot this signal in frequency domain?

S9: Now open Matlab, set the current directory to lab_08, then run the `ReconstructionEPuck.m` file. As in the previous lab, this script loads the signal sampled with the e-puck, performs a linear interpolation and computes the FFT of the interpolated signal. Take a look at the FFT. Are the frequency peaks at the values you expected? In this script is the variable `sampling_frequency` well set? Roughly estimate from your signal its correct value and update it. Run the script again and see if the FFT now corresponds to what you expect.

3 Control a mobile E-puck

3.1 Motor control

The e-puck is a so-called *differential drive* robot which means that just by setting the appropriate speed to the individual motors, the robot can move forward, backward, and even turn on the spot. The commands to set the speed for the motors are:

```
> e_set_speed_left(left_speed)
```

and:

```
> e_set_speed_right(right_speed)
```

The possible values for left_speed and right_speed are between -1000 (full reverse) and 1000 (full forward).

Go to the directory `/e-puck_mobile` and open the file `e-puck_mobile.c` in your favorite text editor. Search the lines 62 and 63.

I10: Modify the values for the wheel speeds to make the robot
1. go forward in a straight line at a medium speed
2. go backward in a straight line at a medium speed
3. turn on the spot clockwise and counter-clockwise
4. go in a circle

For each situation, compile and flash the program to the e-puck, as explained above. Do not forget to add the path to the `EpuckDevelopmentTree/library` in the Makefile before compiling. Run and test your program.

3.2 Obstacle avoidance

In the previous lab, we introduced the Braitenberg robot controller. In this section, you will run the same controller on the e-puck itself. Observe the structure of the controller in *`epuck_mobile.c`*, starting in line 65 (you will find almost the same structure of the last lab).

I11: Change the operation_mode variable, in line 25, to `OBSTACLE_AVOIDANCE`. Compile and flash the program to the e-puck, as explained above. Run your e-puck. Use some obstacles (e.g., you can use your hands) and see whether the e-puck can avoid them.

I12: Now change the Braitenberg parameters in order to make the e-puck to be very reactive on one of the sides, but slow on the other. Compile and flash the program to the e-puck, as explained above.

There are also other ways of programming an obstacle avoidance controller. Here below we explore a simple rule-based (i.e. if “condition_a” satisfied, then do “action_b”) obstacle avoidance algorithm.

I13: Change the code, in the indicated place in the c-file, with a simple rule-based obstacle avoidance algorithm based on the values of the proximity sensors. Once you have finished programming, compile, upload to the e-puck and test your program.