Introduction to the e-puck robot

Xavier Raemy
SWIS group
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Project web page: `http://www.e-puck.org`

November 28, 2006

1 Introduction

The robotic world consists of two main topics: robotic arms (or manipulators) and mobile robots. The e-puck fits in the category of mobile robots.

A mobile robot is a device that is able to move using actuators. Some mobile robots are autonomous, meaning that they take decisions on their own, according to their perception of the environment. A robot that is not autonomous executes a predefined sequence of operations.

The e-puck can be both autonomous or not. It has sensors to perceive the environment and an microcontroller to run a program and move on its own, but it can also just execute a preprogrammed task.

The remainder of this document describes

1. the e-puck hardware (sensors, actuators, communication)
2. the e-puck software, and how to program it

2 The E-puck Hardware

2.1 Microcontroller (MCU)

The central processing unit of the robot (its brain, so to say) is a dsPIC30F6014A microcontroller from Microchip's dsPIC30 family. It is this microcontroller that executes the programs, queries sensors and controls the actuators.

The microcontroller on the e-puck contains a 16-bit RISC CPU that runs at 14.74 MIPS. Almost all instructions use one cycle. It has **no floating point support**, which makes floating point operations very slow (hundreds of instructions for one floating point operation). It also has a DSP core for advanced signal processing.

In addition, the microcontroller contains three types of memory:

- **Flash program memory (144 KB):** This memory space contains your program code, lookup tables and all constants. 144 KB is a lot if you just write code, but it is very limiting if you want add sounds or images. In contrast to your PC, your program runs directly from this memory and is not copied to the RAM for execution.
- **RAM (8192 Bytes):** This is the available RAM for your (dynamic) variables. Variables that you use in your program are automatically stored in here. Note that this memory is volatile, i.e. it is lost when you switch of or reboot the e-puck. When you switch on your e-puck, this memory contains random data.

Microcontroller

From Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Microcontroller

A microcontroller (or MCU) is a computer-on-a-chip (MCU) used to control electronic devices. It is a type of microprocessor emphasizing self-sufficiency and cost-effectiveness, in contrast to a general-purpose microprocessor (the kind used in a PC). A typical microcontroller contains all the memory and interfaces needed for a simple application, whereas a general purpose microprocessor requires additional chips to provide these functions. A microcontroller is a single integrated circuit with the following key features:

- central processing unit
- input/output interfaces such as serial ports
- peripherals such as timers and watchdog circuits
- RAM for data storage
- ROM, EEPROM or Flash memory for program storage
- clock generator - often an oscillator for a quartz timing crystal, resonator or RC circuit

- **EEPROM (4096 Bytes):** This is a non-volatile memory for data that you want to keep even if you switch the robot off. (The robot ID is stored here, for example.) Reading from and writing to this memory is slightly more complicated, though.

The dsPIC30 has dedicated peripherals embedded on the same chip. Those in use on the e-puck are:

- **Five timers:** The timers are counters that generate hardware interrupts periodically.
- **Analog-to-Digital Converter (12-bit, 100 Ksps, 16 channels):** The ADC can measure a voltage on a specific pin of the chip and convert it to a 12-bit value. Up to 100000 samples per second can be acquired. This is used for the infrared sensors and the accelerometer (see below).
- **Two UART modules:** UART is the 0 to 5V version of the -10 to +10V RS-232 serial port that you find on your PC. (Some of the new PCs don't have this port any more.)
- **I^2C:** The I^2C bus is a addressable serial bus that is used to communicate with the extension modules.
- **I^2S:** The I^2S bus is an audio serial bus that is used to transmit audio data to an audio codec chip.

All theses hardware extensions can trigger interrupts. Interrupts are often used when you want to react quickly to a hardware request. If an interrupt occurs, the CPU pauses the execution of the main program, execute the interrupt service routine (ISR) and returns to main program at the exact point where it leaved it. This is transparent for the main program. For instance, on the e-puck robot, timer interrupts are used by the library to control the motors signals timings.

2.2 Motors and Locomotion

The e-puck robot is a *differential drive* robot with two independently driven wheels on the same axis. In order to maintain the stability, the plastic body touches the ground in two points. Hence,

the e-puck can only be used on flat surfaces with low friction - such as your desk, for example.

Each wheel is driven by a stepper motor. A stepper motor is an incremental motor, i.e. it can only move forwards and backwards in discrete steps. Such motors are driven with two square wave signals (4 wires), shifted by a quarter of a period. Each change of one of the signals is a step. The wheel speed is therefore proportional to the frequency of these signals. The e-puck library contains functions that produce these signals for a given speed.

As each motor step has the same length, you can estimate the distance traveled by counting the number of motor steps. However, you only know the number of steps that the motors were supposed to turn, not the number of steps they effectively did (as you would get with encoders, for instance).

Further reading
Wikipedia: Stepper motor, http://en.wikipedia.org/wiki/Stepper_motor

2.3 Sensors

The e-puck robot has four types of sensors that allow it to perceive the environment:

- 8 infrared sensors
- 1 accelerometer (3D)
- 3 microphones
- 1 color camera (640 x 480 pixels)

When working with these sensors, keep in mind that we are in the real world. All sensors are different because of the manufacturing process, aging, temperature and light conditions, and so on. In addition, the measured signals contain measurement noise (coming from the physics of the sensor), electrical noise and possibly interference. You cannot get rid of these issues, but have to deal with them.

2.3.1 Infrared Sensors

The e-puck robot is equipped with 8 infrared sensors (Vishay TCRT1000). Theses sensors consist of an infrared light emitting diode (IR-LED) and a photo transistor sensitive to infrared light. The photo transistor is connected to the ADC, so that we can at any time get a digital value of the IR light that the sensor perceives. The IR-LED can simply be switched on and off.

The infrared sensors can be used for ambient infrared light measurements and for obstacle proximity measurements.

Ambient light measurements are taken by simply measuring the amount if IR light with the photo transistor when the IR-LED is switched off. Note that there are many sources that emit light in the infrared spectrum. The brightest among them is the sun.

Proximity measurements indicate the proximity of the sensor to an obstacle. To measure this, the following steps are performed:

1. Measure the output of the photo transistor (value A).
2. Switch on the IR-LED.
3. Wait several microseconds until the IR-LED reaches its full power.
4. Measure the output of the photo transistor again (value B).

5. Switch off the IR-LED.
6. Wait several microseconds that the IR-LED is off.
7. Return B-A, the difference between the two measurements.

The value we refer to as the "proximity" is in fact just a difference between two measurements: one with the IR-LED off and one with the IR-LED on. If there are no obstacles, there is no difference between these two values and 0 (or a value close to 0) is returned. If there is an obstacle, some of the light (energy) that the IR-LED is emitting will be reflected. Therefore, B will be bigger than A and the return value bigger than 0.

2.3.2 Accelerometer

The e-puck has a 3D accelerometer (Linear Technologies, A7260), located close to the right wheel. The chip has three analog outputs that are proportional to the measured acceleration in x, y and z direction. These analog outputs are connected to the ADC of the microcontroller.

2.3.3 Microphones

The microphones are a little bit more difficult to use. They are preamplified, and the output of each microphone is connected to an ADC input of the microcontroller.

The e-puck library contains functions to obtain individual samples from the microphones. However, if you want to use the microphones more seriously, you need to study the dsPIC30 datasheet and configure the ADC module by yourself.

2.3.4 Camera

The robot is equipped with a tiny 640 x 480 color camera, located in front of the e-puck.

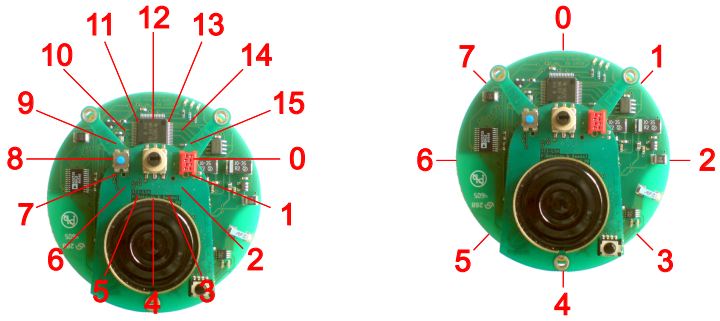
Note that the microcontroller doesn't have enough memory to store an image of this size. Therefore, you can only use a small part of the image (roughly 40 x 40 pixels) at the same time.

The camera uses a lot of resources available on the chip and your application need to take that into account. For more information, please have a look at the source code of the library.

2.4 User Interface

The robot has several ways to interact with you:

- **Power switch and LED:** The power switch is located at the back of the right wheel. Pull the switch down to switch on the robot. You'll see a green light appearing above this switch.
- **Reset button:** The little blue push button on the top of the robot reboots the robot when you press it.
- **Selector:** This is the small black stick on the top of the robot. You can turn it to one of 16 positions, as shown in Figure 1 (a).
- **LED ring:** The 8 red LEDs around the robot can be independently switched on and off. Figure 1 (b) shows the numbering of the LEDs.
- **Body LED:** The body led illuminates the plastic body of the robot in green.
- **Front LED:** This is a red high power LED in front of the robot.



(a) (b)

Figure 1: (a) The 16 selector positions. (b) Numbering of the LEDs on the LED ring.

2.5 Communication

There are two ways to communicate with an e-puck from your computer:

1. The robot is equipped with Bluetooth. You need to establish the connection from the PC (the robot is always slave). Once connected, the Bluetooth connection emulates a serial cable, transparently to both the robot and the PC. This is the most convenient way to communicate with the robot.
2. The robot has an RS-232 connector (-10V to +10V) which allows to connect it to a PC serial port via a cable. Note that you need a serial cable with a micromatch connector for this.

2.6 Battery

The battery is located between the two wheels. Note that batteries cannot be charged inside the robot. If empty, replace the battery with new one and charge the drained battery with the external charger.

3 The E-puck Software

You can write programs in C or directly in assembler. For convenience and implementation speed reasons, you'll almost always use C unless you really need to do fancy things (e.g. special signal processing instructions).

3.1 Files and Libraries

Download and install the C30 compiler from Microchip. Detailed instructions for this are available on http://www.e-puck.org.

The installation of the development tools will give you a set of files which we will refer to as *the development tree*. Assuming you have extracted the archive in `${epuckroot}`, you will find the library in `${epuckroot}/e_puck/library`. Here, you'll find several subdirectories that contain the header files (*.h) of the standard e-puck library. The functions and descriptions in these files should be enough for you to use the library.

Here is a brief overview of the subdirectories:

- `a_d`

This directory contains the files needed to use the ADC. These are low-level functions that you usually don't need to access directly - except if you want to do fancy things with the ADC.

- `camera`

  This directory contains the file needed to access the camera.

- `codec`

  This directory contains the file needed to play sounds with the loudspeaker.

- `I2C`

  This directory contains the files needed to use the I2C bus. Unless you want to have full control of the I2C bus, you don't need to include theses files or use theses functions. This is a very low level layer that is use by higher level functions of extension module, like the Zigbee communication module.

- `motor_led`

  This directory contains the file needed to initialize the I/O ports of the microcontroller, define constants used by all other module. It also contains functions to use the motors and control the LEDs.

- `uart`

  This directory contains the file needed to communicate via UART (serial ports: RS-232 or Bluetooth).

- `std_microchip`

  This directory contains the definitions and the sources of all the very low level standard functions (e.g. `sprintf`).

- `contrib`

  This directory contains subdirectories with all files needed to use the hardware extensions of the robot.

- `bluetooth`

  The files in this directory are needed only if you want to use the robot as a Bluetooth **master**. As mentioned before, the Bluetooth module is a slave by default (the PC initiates the connection).

It is often simpler to begin with a template that works than beginning from scratch. The best way to understand how things work is to read to source code of the example programs. If you want to know the details about a function, have a look at its source code in the e-puck library.

3.2 General Considerations when Programming the Robot

When you program the e-puck robot, you are very close to the hardware. This means you have to keep in mind several things:

- There is **no operating system** on the robot. The robot will reset and directly run you code[1]. Hence, your program is responsible of everything, from the definition of every pin (is it an output or an input pin?) to the configuration of the embedded hardware (e.g. ADC). For the standard things, the e-puck library will take care of this.
- There is **only one process** running (`fork` doesn't exist). You can't have multiple threads neither.

[1] In fact the robot will first run the bootloader and then execute your program. See "Tutorial for programming the e-puck robot in Linux" for more informations.

- There is **no memory management**. You cannot allocate or free memory at run time (`malloc` and `free` don't exist).
- There is **no floating point support** on the CPU. The compiler allows you to use floats, but these operations will be emulated with hundreds of integer instructions. Hence, this noticeably slows down your program. You should avoid using floating point operations and use integer or fixed point operations instead.
- You have a very **limited amount of program memory** available. It is usually enough for your executed code, but very restrictive if you want add sounds or images to your program.
- You have a very **small amount of RAM** available for your variables. Hence, you cannot use large arrays (big images, sounds). Always think about optimizing your algorithms in terms of memory usage.