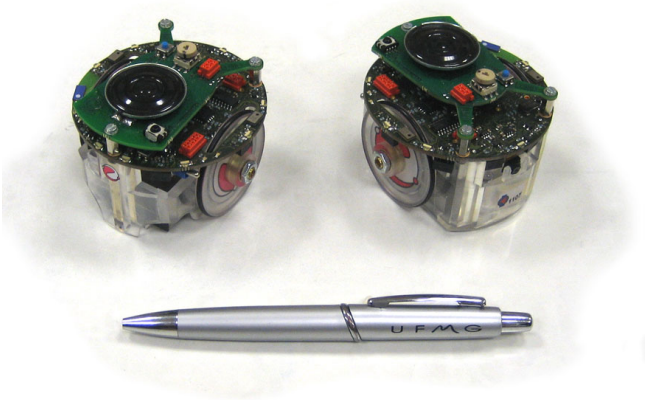


E-puck Tutorial

Renato Florentino Garcia

Laboratório de Visão Computacional e Robótica - VeRLab
Universidade Federal de Minas Gerais - UFMG

20 de setembro de 2008

Sumário

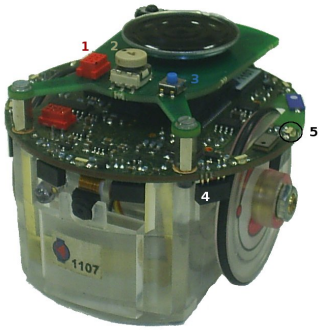


Figura 1: E-puck e indicação de alguns componentes.

1 Introdução

O objetivo deste tutorial é explicar como utilizar e criar programas para um robô e-puck usando a biblioteca de software padrão do projeto, e como a partir dos fontes do driver para o Player compilá-lo e usá-lo.

Para programar um e-puck é necessário ter um computador equipado com algum dispositivo bluetooth, pois os programas desenvolvidos serão enviados ao robô utilizando-se o bluetooth.

Nos exemplos mostrados os comandos que devem ser feitos do lado do computador são para o ambiente Linux, caso se esteja utilizando o Windows existe um outro tutorial que se encontra disponível tanto no site oficial do projeto e-puck [2] quanto no site do VeRLab [5]. Neste tutorial, escrito por Francesco Mondada e Michael Bonani, da Ecole Polytechnique Fédérale de Lausanne (EPFL), está explicado como o mesmo resultado do ser obtido no Windows. A distribuição Linux usada na construção dos exemplos foi a Ubuntu 8.04, mas o mesmo procedimento poderá ser feito em outras distribuições com pouca ou nenhuma alteração.

2 O robô

O e-puck(figura 1) é um robô diferencial de pequeno porte que possui vários sensores e atuadores. Alguns de seus componentes são: uma câmera com resolução de 640x480, oito sensores infra-vermelho, dez LEDs, três microfones, um auto-falante, um acelerômetro 3D, dois motores de passo, e duas portas seriais, sendo que uma delas está ligada a um dispositivo bluetooth. Ele é embarcado com um dsPIC modelo 30F6014A, que é programável por meio da interface bluetooth.

Cada e-puck possui um número de identificação de quatro dígitos. Este número é único, e geralmente pode ser lido em um papel adesivo que se encontra colado em algum local visível do mesmo. Este mesmo número de identificação também é encontrado no nome do dispositivo bluetooth representado pelo robô, que aparece sempre no seguinte formato: e-puck_XXXX, onde "XXXX" deve ser substituído pelo próprio número de identificação.

O projeto e-puck é aberto, tanto a especificação do hardware quanto a biblioteca de software que fornece acesso a este hardware. Todo o projeto está disponível sob uma licença[1] que permite a qualquer pessoa montar um exemplar, modificá-lo e/ou redistribuí-lo. No site do projeto[2] estão disponíveis para download os esquemáticos para a montagem do hardware, a biblioteca oficial, e alguns outros tutoriais e exemplos.

3 Compilando um programa

Os programas feitos para um e-puck podem ser escritos na linguagem C ou em assembly, porém, por causa da facilidade de se programar em C quando comparado ao assembly, além de toda a interface da biblioteca oficial do projeto estar codificada em C, todos os exemplos e programas neste tutorial também serão escritos nesta linguagem.

O compilador utilizado será o MPLABC30, versão 3.01 e edição de estudante, que pode ser obtido gratuitamente no site da Microship[3]. Apesar de ser nativo para Windows esse compilador funciona muito bem no Linux se executado pelo Wine.

No site oficial do projeto e-puck[2] é possível fazer o download de uma cópia da biblioteca oficial do projeto, juntamente com alguns programas úteis para o desenvolvimento na plataforma. Quando este tutorial estava sendo feito a versão mais atual datava de 29/02/08, e por ser mais prático foi escolhida a opção que continha os fontes e os binários já compilados da mesma.

Para facilitar o processo de compilação e integrar o MPLABC30 ao make, foram criados dois scripts: compile.py e linker.py, ambos podem ser encontrados no site do VeRLab[5]. Antes de compilar algum programa pela primeira vez, deve ser configurado o script linker.py. À variável MPLAB_C30_PATH deve ser atribuído o caminho absoluto da pasta onde o MPLABC30 foi instalado. Da mesma forma, à variável STD_LIB_DIR deverá ser atribuído o caminho absoluto da pasta onde se encontram os arquivos compilados da biblioteca oficial do projeto e-puck.

Os fontes do projeto a ser compilado podem estar em um único diretório base ou distribuídos em subdiretórios. Seja qual for a configuração, para compilá-lo o script compile.py deverá ser executado a partir do diretório base. Caso os arquivos dentro de algum subdiretório devam ser compilados, o nome deste subdiretório deve ser dado como argumento para o compile.py.

Onde houver arquivos fontes a serem compilados, tanto no diretório base quando nos subdiretórios, deverá haver também um Makefile indicando como compilar estes arquivos. Além do Makefile, deverá existir um arquivo Linkfile listando as libs da biblioteca padrão do e-puck que deverão ser utilizadas. Por exemplo, se o projeto utiliza alguma função para acender os LEDs, a libmotor_LED.a deverá ser ligada aos arquivos objeto obtidos na compilação. Para que isso seja feito, em uma linha do arquivo Linkfile deverá estar escrito motor_LED.

No site do VeRLab [5] se encontra um projeto com todos estes arquivos para exemplificar o processo de compilação. Dentre eles está o fonte leds.c que é o programa que será compilado como exemplo, ele também está listado no apêndice A.1. O nome do arquivo com o projeto é projetoExemplo.tar.gz, e após ser descompactado, no diretório base estarão os scripts compile.py e linker.py,

além da pasta chamada leds. Para compilá-lo, após atribuir o valor correto às variáveis MPLAB_C30_PATH e STD_LIB_DIR do script linker.py, basta executar o comando:

```
$ ./compile.py leds
```

Como resultado da compilação deverão ter sido criados três arquivos no diretório base: output.cof, output.hex e proj.map.

4 Carregando um programa compilado no e-Puck

Uma vez que o projeto tenha sido compilado, o resultado obtido será um arquivo .hex, que deverá ser carregado na memória do e-puck. Uma informação que será necessária é o numero de identificação do e-puck, que também será descrito como pode ser obtido.

O programa utilizado para copiar o arquivo .hex para a memória do e-puck será o epuckuploadbt, cujo código fonte é distribuído junto da biblioteca oficial do projeto.

Caso o número de identificação do e-puck não esteja colado no mesmo, pode-se descobri-lo entrando com o seguinte comando em um terminal:

```
$ hcitool scan
```

Se houver algum e-puck ligado e ao alcance do sinal bluetooth do computador, a saída deverá ser parecida com a seguinte:

```
Scanning ...
10:00:E8:52:A9:69       e-puck_1107
```

Neste caso o nome do e-puck é e-puck_1107, donde se tira que o número de identificação único do mesmo é 1107.

Para carregar o arquivo .hex, neste exemplo com o nome output.hex, o epuckuploadbt deverá ser executado tendo como argumento o arquivo a ser carregado e o número do e-puck. Com o nome do arquivo .hex e o número do e-puck dados acima, o comando seria:

```
$ epuckuploadbt output.hex 1107
```

O programa pedirá que se resete o robô, o que é feito apertando-se o botão que se encontra na parte superior de mesmo, e que pode visto próximo ao número 3 na figura 1.

5 O hardware e a biblioteca do projeto e-puck

Todo o acesso ao hardware do e-puck pode ser feito através da biblioteca oficial do projeto, utilizando os cabeçalhos C e as libs compiladas. A documentação da biblioteca oficial pode ser gerada a partir dos comentários nos fontes usando o programa doxygen, ou pode ser obtida em formato html do site do VeRLab[5].

Todo programa feito para um e-puck deve ter uma função main, que é de onde o código começará a ser executado. Porém, como o programa será o único na memória do robô, assim que a função main terminar ela será novamente chamada, e assim indefinidamente.

Quando se carrega um programa na memória do e-puck é preciso que ele seja resetado no processo, porém, de acordo com o código que estiver sendo executado no momento o reset pode falhar, e terá que ser repetido até que funcione. Isto não acarreta maiores problemas, mas é um pouco inconveniente. Uma solução encontrada foi colocar um loop infinito antes do fim da função main, como pode ser visto no código fonte mostrado no apêndice A.1, uma vez estando dentro do loop o reset funcionará sem problemas. Para o caso de programas mais complexos, onde não se alcançará o fim da função main, logo no inicio do programa pode ser colocada uma condição que entrará em um loop infinito para uma dada posição da chave seletora. Como trabalhar com a chave seletora é explicado na subseção 5.3.

5.1 Motores

Um e-puck é equipado com um motor de passo para cada uma de suas duas rodas. O cabeçalho da biblioteca oficial que descreve o acesso a esses motores é o e_motors.h, e existem três cabeçalhos diferentes, em cada um deles os motores serão controlados pelos timers do dsPIC de forma diferente.

Os motores funcionam com vinte passos por revolução, e são ligados às rodas por uma redução $50:1$, portanto, a cada 1000 passos do motor as rodas descrevem um giro completo. As velocidades dos motores são atribuídas pelas funções e_set_speed_left e e_set_speed_right, cujo argumento diz a quantos passos por segundo os motores devem girar. A velocidade máxima dos motores é de 1000 passos por segundo, o que equivale à uma velocidade angular de $2\pi rad/s$ nas rodas.

As funções e_get_steps_left e e_get_steps_right retornam a quantidade de passos que já foi dada por cada motor. Já as funções e_set_steps_left e e_set_steps_right atribuem um valor à quantidade de passos já dados. Isso pode ser útil para implementar um sistema de hodometria por exemplo.

5.2 LEDs

Os e-pucks possuem oito LEDs distribuídos radialmente (um deles pode ser visto junto ao número 5 na figura 1), um na base, e outro na frente. Eles são úteis para serem usados no processo de debug, ou como indicadores de algum estado interno. Os LEDs são acessados pelo cabeçalho e_led.h, e são ligados pela mesma lib usada pelos motores.

5.3 Chave seletora

Outro dispositivo presente nos e-pucks é uma chave seletora de 16 estados, item próximo ao número 2 na figura 1. O estado da chave fornece uma flag que pode ser usado para qualquer propósito dentro de algum programa. Para obter um inteiro representando a atual posição basta calcular a seguinte equação:

$$SELECTOR0 + 2 \cdot SELECTOR1 + 4 \cdot SELECTOR2 + 8 \cdot SELECTOR3.$$

Onde SELECTOR0, SELECTOR1, SELECTOR2 e SELECTOR3, são macros definidas no cabeçalho e_epuck_ports.h.

5.4 Portas seriais e bluetooth

Há em cada e-puck duas interfaces seriais UART, UART1 e UART2. Estas interfaces podem ser acessadas usando o cabeçalho e_uart_char.h.

A interface UART1 está ligada ao dispositivo bluetooth, porém não existe nenhuma diferença no código que deve ser feito para acessá-la quando comparada com a UART2. Caso se use a UART1 em conjunto com o bluetooth, o computador com o qual se comunica deverá estar também equipado com um dispositivo bluetooth, e do seu ponto de vista os dados transmitidos podem ser tratados como aqueles provenientes de uma conexão serial RS-232 usando fios. Já a UART2 está ligada a um conector que se encontra na parte superior do e-puck, que pode ser visto junto ao número 1 na figura 1, e é necessário usar fios como o meio físico para o transporte dos dados.

5.5 Sensor infravermelho de proximidade

Estão distribuídos ao longo do perímetro do corpo dos e-pucks oito sensores infravermelhos para detectar algum objeto que esteja próximo, um deles pode ser visto junto ao número 4 na figura 1. O cabeçalho da biblioteca oficial que os descreve é o e_prox.h.

A qualidade do sensor infravermelho não é boa, olhando um único sensor há uma grande variação quando o e-puck se inclina para trás e para frente durante o deslocamento; e quando se compara os vários sensores entre si, novamente há uma grande variação entre os valores lidos para uma mesma distância. Além do mais, para distâncias maiores que $3cm$ a diferença entre as leituras dadas pelos sensores em função da distância é muito pequena, sendo difícil fazer uma estimativa confiável. Na seção A.2 estão alguns gráficos mostrando estes valores.

6 Comunicação computador/e-puck

A forma mais prática de comunicação entre os e-pucks e um computador é através dos módulos UART. Cada e-puck possui dois controladores UART, UART1 e UART2, sendo que o UART1 está conectado ao módulo bluetooth. Do lado do computador, é possível associar os dados recebidos pelo módulo bluetooth à uma interface serial, e desta forma cria-se um canal de comunicação sem fios entre um e-puck e um computador. Uma segunda possibilidade é usar a UART2, que é acessível via um conector que utiliza o padrão RS-232, esse conector pode ser visto na indicação de número 1 da figura 1. Esta segunda opção exige o uso de fios como meio físico para a transmissão dos dados.

Pela inconveniência de se ter que lidar com fios conectando o e-puck ao computador, será utilizado o UART1 juntamente com o módulo bluetooth para que a comunicação possa ser feita sem fios. Do lado do e-puck, o envio e recebimento de mensagens é feito usando as funções definidas no cabeçalho e_uart_char.h, pertencente à biblioteca oficial do projeto. No programa feito para o e-puck não existirá nenhuma diferença entre uma comunicação serial com ou sem fios. Do lado do computador, é necessário associar o bluetooth a um dispositivo serial, e toda a comunicação será feita como se houvesse uma ligação serial comum por meio de fios.

O primeiro passo a ser feito é descobrir o endereço bluetooth do e-puck que se quer trabalhar. Com um terminal aberto, entra-se com o seguinte comando:

```
$ hcitool scan
```

Se houver algum e-puck ligado e ao alcance do sinal bluetooth do computador, a saída deverá ser parecida com a seguinte:

```
Scanning ...
10:00:E8:52:A9:69       e-puck_1107
```

Neste caso específico o endereço é 10:00:E8:52:A9:69. Outra informação disponível é o nome do e-puck: e-puck_1107, donde se tira que o número de identificação único do mesmo é 1107.

Uma vez tendo-se o endereço, chama-se o programa rfcomm, que emulará uma porta serial RS-232 no computador. A sintaxe do comando é a seguinte:

```
$ rfcomm connect <dev> [bdaddr] [channel]
```

Onde campo [bdaddr] é o endereço bluetooth do e-puck, o campo <dev> especifica o device no qual a porta serial será emulada, e [channel] qual o canal rfcomm que é usado pelos e-pucks. A porta serial será mapeada em /dev/rfcommX, onde X é o número passado em <dev>. Já a opção [channel], para a comunicação com os e-pucks deverá ser sempre 1, que é a opção padrão, e podendo portanto ser omitida. Caso se queira criar uma porta serial em /dev/rfcomm0, e que esteja ligada com o e-puck encontrado pelo comando "hcitool scan" feito acima, o comando seria o seguinte:

```
$ rfcomm connect 0 10:00:E8:52:A9:69
```

Neste ponto, pode-se usar o device criado como se fosse uma porta serial física tradicional.

7 Player

Uma das formas de se construir programas para controlar os e-pucks é através da biblioteca disponível no site oficial do projeto e-puck, como já foi visto até agora. Essa opção apresenta algumas desvantagens, entre elas estão o baixo poder de processamento de um dsPIC quando comparado a um PC, a única linguagem de programação disponível é C, e o código dos programas que forem feitos não será portável entre outros robôs.

Uma alternativa que contorna boa parte destes problemas é construir o programa usando a biblioteca Player[4]. O Player implementa uma camada de abstração entre o programa desenvolvido e o robô, e com isso a programação é feita para um robô genérico, não se tem mais a preocupação em controlar uma arquitetura de hardware específica. Além disso, oficialmente o projeto disponibiliza bibliotecas para a programação em C, C++, e Python. O Player está disponível sob a licença GPL, e roda somente em sistemas operacionais compatíveis com o padrão POSIX; que inclui o Linux, mas que exclui o Windows.

Não existe oficialmente o driver para que um e-puck funcione com o Player, mas um está sendo desenvolvido no VeRLab[5], e pode ser usado para este propósito. Como compilá-lo e usá-lo será explicado a seguir.

7.1 Compilando e instalando o driver

No site do VeRLab é possível baixar o pacote .tar.gz com o código do driver. Uma vez com o pacote no computador, descompacte-o e de um terminal entre na pasta criada. Digite então:

```
$ ./configure
$ make
```

Agora a biblioteca com o driver já deve estar compilada. Para instalá-la existem duas alternativas, a primeira é fazer a instalação no sistema, e a segunda é instalar a biblioteca em um diretório local. Como se está instalando a biblioteca diretamente, sem se criar um pacote no formato oficial da distribuição Linux qualquer que se esteja usando, é recomendável instalar em um diretório local para que seja mais simples uma possível desinstalação. Para isso basta digitar o seguinte comando, estando com o terminal ainda dentro da pasta do driver:

```
$ make install DESTDIR=<caminho>
```

Onde <caminho> é o local onde o driver será instalado, por exemplo ∼/driverEpuck, que indicará uma instalação em um diretório chamado driverEpuck, dentro do home do usuário.

7.2 Usando o driver

O arquivo de configuração do Player que deverá ser usado com o driver do e-puck deverá ser parecido com o abaixo:

```
driver
(
  name "epuck"
  plugin "libepuck"
  provides ["position2d:0" "ir:0"]
  port "/dev/rfcomm0"
)
```

Onde a opção port indica qual será a porta serial em que o e-puck está conectado, e não pode ser omitida.

Quando se for executar o programa Player, o local onde a biblioteca com o driver do e-puck foi instalada deve ser de conhecimento do sistema, sendo que para isso basta exportar uma variável de ambiente com essa informação. Seguindo o exemplo mostrado na subseção 7.1, o comando seria o seguinte:

```
$ export LD_LIBRARY_PATH='/home/<usuário>/driverEpuck/usr/local/lib'
```

Onde <usuário> representa o nome da pasta pessoal do usuário em questão.

Caso a biblioteca tenha sido instalada em alguma pasta que já esteja no path do sistema este último passo não é necessário.

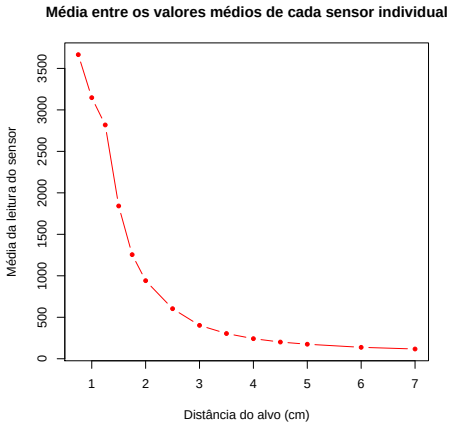
Agora o Player já deverá estar sendo executado, e o ambiente já está preparado para executar os programas desenvolvidos.

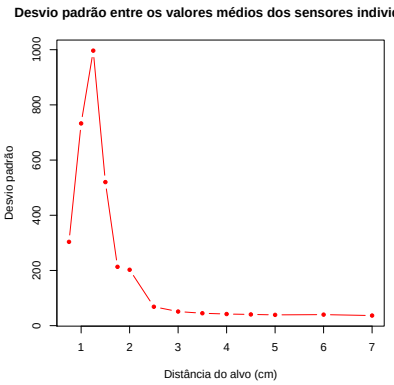
A Apêndices

A.1 Leds.c

```
include <p30f6014A.h>

#include <motor_led/e_epuck_ports.h>
#include <motor_led/e_init_port.h>
#include <motor_led/e_led.h>

#define LED_ON 1

int getselector() {
return SELECTOR0 + 2*SELECTOR1 + 4*SELECTOR2 + 8*SELECTOR3;
}

int main()
{
  int selector;

  e_init_port();

  //Reset if Power on (some problem for few robots)
  if (RCONbits.POR)
  {
    RCONbits.POR=0;
    __asm__ volatile("reset");
  }

  selector = getselector();

  if(selector == 0)
  {
    e_set_led(0, LED_ON);
  }
  else
  {
    e_set_led(1, LED_ON);
  }

  while(1);
  return 0;
}
```

A.2 Leituras dos sensores infravermelhos





Referências

[1] Licença do projeto e-puck. http://www.e-puck.org/index.php?option=com_content&task=view&id=18&Itemid=45.

[2] Página oficial do projeto e-puck. http://www.e-puck.org.

[3] Página microship. http://www.microchip.com.

[4] Página do projeto player. http://playerstage.sourceforge.net.

[5] Projeto e-puck player driver do verlab. http://www.verlab.dcc.ufmg.br/doku.php?id=projetos:epuck-player:index.