OpenCV, RaspiCam e WiringPi

1 Introdução

Iremos desenvolver o nosso projeto em C++. A linguagem padrão escolhida pela Raspberry não é C++ mas Python, provavelmente pela facilidade de aprendizagem. Porém, os testes mostram que C++ é algo como 10 a 100 vezes mais rápido do que Python:

```
http://benchmarksgame.alioth.debian.org/u64q/compare.php?
lang=python3&lang2=gpp
```

Ou seja, não é possível desenvolver sistemas "sérios" em Python (principalmente os de tempo real). Pelo fato de C++ não ser a linguagem "oficial" da Raspberry, não há funções em C++ disponibilizadas oficialmente para controlar alguns periféricos do Raspberry, entre elas capturar imagem da câmera e para controlar GPIO. Assim, utilizaremos duas excelentes bibliotecas de desenvolvedores independentes: raspiCam e wiringPi.

2 C++, OpenCV e Cekeikon

Instale C++, OpenCV e Cekeikon seguindo os passos descritos em:

```
http://www.lps.usp.br/hae/software/cekeikon5.html
```

Nota: O uso de Cekeikon é opcional. Não há problema em fazer o projeto sem esta biblioteca.
Compile e execute programas C++, OpenCV e Cekeikon. Para isso, vá para o diretório:

```
cd ~/cekeikon5/cekeikon/samples/crt
```

E compile:

```
$ compila hello.cpp
$ compila hello_opencv -ocv
$ compila hello_cekeikon -c
```

Depois execute-os:

```
$ ./hello
$ ./hello_opencv
$ ./hello_cekeikon
```

Nota: Se você não quer ficar escrevendo ./ toda hora, faça a alteração sugerida na apostila "raspberry.odt".

3 RaspiCam

Muitos modelos de webcam são incompatíveis com Raspberry:

```
http://elinux.org/RPi_USB_Webcams
```

Assim, vamos usar a câmera própria para Raspberry (versão 1, de 5 Mpixels), que com certeza é compatível. Só que a forma de acessar câmera da Raspberry é diferente do acesso a webcam, disponível em OpenCV. Para acessar câmera da Raspberry, vamos usar a biblioteca Raspicam. Instale Raspicam seguindo os passos descritos em:

```
https://www.uco.es/investiga/grupos/ava/node/40
```

Esta apostila supõe o uso da versão 0.1.4. Teste compilar e executar os programas exemplos que acompanham esta biblioteca. Leia os códigos.

Fiz um “encapsulamento” para facilitar (ainda mais) o uso da biblioteca Raspicam. É o arquivo cekraspicam.h:

```
// cekraspicam.h
#include <raspicam/raspicam_cv.h>
using namespace raspicam;
class CEKRASPICAM {
public:
  RaspiCam_Cv cam;
  CEKRASPICAM(int nl=480, int nc=640, bool colorido=true) {
    if (colorido) cam.set(CV_CAP_PROP_FORMAT, CV_8UC3);
    else cam.set(CV_CAP_PROP_FORMAT, CV_8UC1);
    cam.set(CV_CAP_PROP_FRAME_HEIGHT, nl);
    cam.set(CV_CAP_PROP_FRAME_WIDTH,  nc);
    if (!cam.open()) erro("Error opening the camera");
  }
  CEKRASPICAM& operator>>(Mat_<COR>& image) {
    cam.grab();
    cam.retrieve(image);
    return *this;
  }
  CEKRASPICAM& operator>>(Mat_<GRY>& image) {
    cam.grab();
    cam.retrieve(image);
    return *this;
  }
  void set(int propId, double value) {
    cam.set(propId,value);
  }
  ~CEKRASPICAM() { cam.release(); }
};
```

Veja abaixo o programa-exemplo cekraspicam.cpp que usa cekraspicam.h para capturar imagens da câmera e mostra-as na tela.

```
// cekraspicam.cpp
// Exemplo de captura de imagem colorida do raspicam
// compila cekraspicam -c -r
#include <cekeikon.h>
#include <cekraspicam.h>
int main (int argc, char **argv) {
  CEKRASPICAM cam;
  Mat_<COR> image;
  namedWindow("janela");
  int ch=-1;
  while (ch<0) {
    cam >> image;
    imshow("janela",image);
    ch=waitKey(30);
  }
}
```

Você deve compilá-lo e executá-lo na Raspberry com:

```
$compila cekraspicam -c -r
```

Este programa captura imagens 480x640 pixels. Se quiser capturar em outra resolução (por exemplo, 240x320), escreva:

```
CEKRASPICAM cam(240,320);
```

4 Driver Motor Ponte H L298n

Este driver já foi utilizada no projeto anterior. Há alguma controvérsia sobre como devem ser feitas as ligações de alimentação e dos jumpers da Ponte-H:

`http://blog.filipeflop.com/motores-e-servos/motor-dc-arduino-ponte-h-l298n.html`

`http://www.instructables.com/id/Control-DC-and-stepper-motors-with-L298N-Dual-Moto/`

O que funcionou foi colocar a alimentação entre os conectores 4 (+5V) e 5 (0V), e manter o jumper 3 no lugar (figuras 5 e 6). Nada foi ligado no conector 6.

Os conectores 1 e 2 devem alimentar o motor A e 13 e 14 o motor B (figura 6).

Os conectores 8 e 9 são as entradas dos sinais para acionar o motor A e 10 e 11 do motor B. O motor A vai girar de acordo com a tabela 1.

Tabela 1: Sinal de entrada do ponte-H e o sentido de rotação do motor.

Motor A	conector 8	conector 9
horário	5V	GND
anti-horário	GND	5V
ponto morto	GND	GND
freio	5V	5V

O mesmo esquema é aplicado aos conectores 10 e 11 para controlar o motor B.

Se quisesse usar PWM por hardware, os jumpers 7 e 12 (figura 6) deveriam ser retirados e os sinais de PWM deveriam ser injetados nos pinos 7 e 12. Como vamos usar PWM por software, mantenha esses jumpers nos seus lugares.

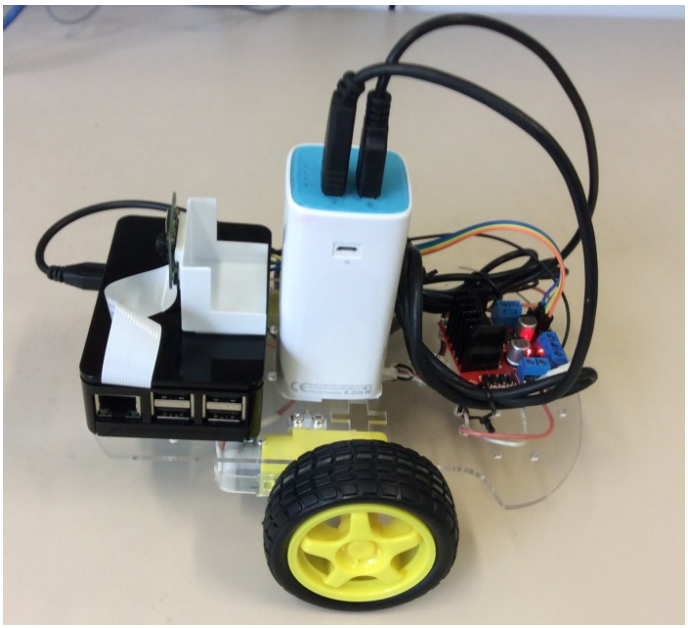

Figura 4: Carrinho mostrando as conexões de alimentação: 2A para Raspberry e 1A para ponte-H e os motores.

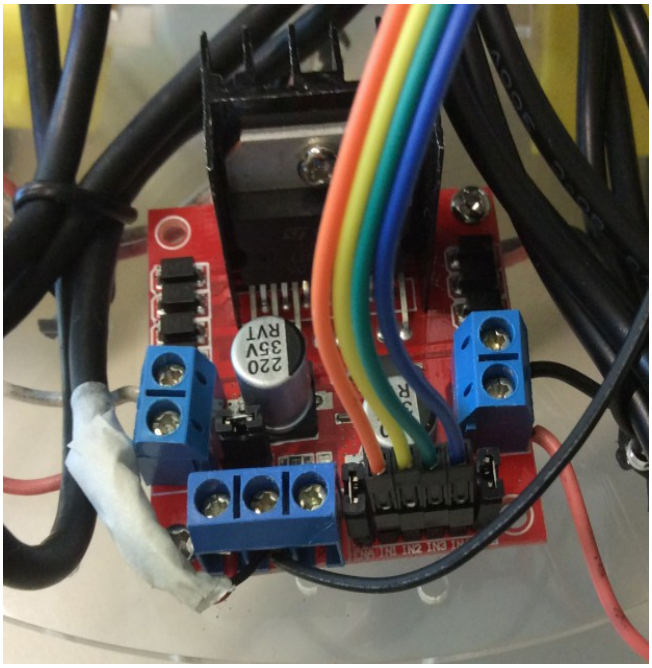

Figura 5: Ponte H L298n para Arduino montado no carrinho.

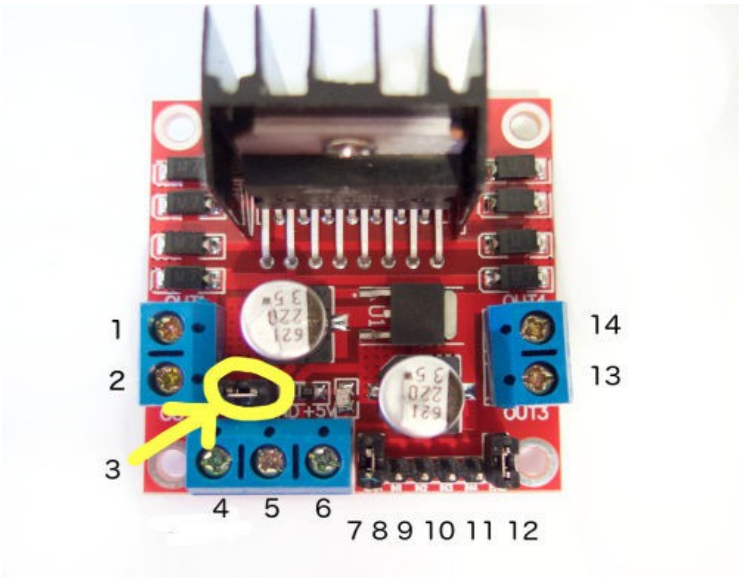

Figura 6: Ligações da ponte H L298N (do instructables.com).

5 WiringPi

Raspberry Pi 3 possui 40 pinos, dos quais 26 são GPIO (figura 3).

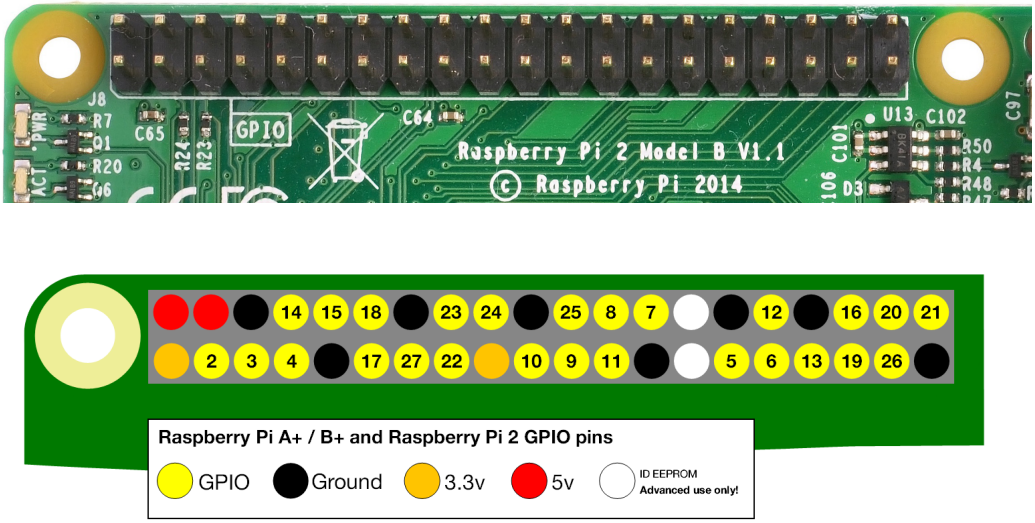

Figura 3: Pinagem de GPIO da Raspberry (do site de Raspberry).

Utilizaremos 4 pinos GPIO como saída, para controlar os dois motores do carrinho.

WiringPi uma é biblioteca para acessar GPIO escrita em C para Raspberry Pi.

```
http://wiringpi.com/
```

Entre as características interessantes, está "software PWM": gera PWM via software, sem hardware extra.

Instale WiringPi conforme:

```
http://wiringpi.com/download-and-install/
```

Instalei de acordo com "Plan B", a versão e687f3f de 2017-01-24.

WiringPi adota uma numeração de pinagem diferente do Raspberry (tabela 2).

Tabela 2: Numeração de pinagem do WiringPi.

BCM	wPi	Name	Physical	Name	wPi	BCM
		3.3v	1 || 2	5v		
2	8	SDA.1	3 || 4	5v		
3	9	SCL.1	5 || 6	0v		
4	7	GPIO. 7	7 || 8	TxD	15	14
		0v	9 || 10	RxD	16	15
17	0	GPIO. 0	11 || 12	GPIO. 1	1	18
27	2	GPIO. 2	13 || 14	0v		
22	3	GPIO. 3	15 || 16	GPIO. 4	4	23
		3.3v	17 || 18	GPIO. 5	5	24
10	12	MOSI	19 || 20	0v		
9	13	MISO	21 || 22	GPIO. 6	6	25
11	14	SCLK	23 || 24	CE0	10	8
		0v	25 || 26	CE1	11	7
0	30	SDA.0	27 || 28	SCL.0	31	1
5	21	GPIO.21	29 || 30	0v		
6	22	GPIO.22	31 || 32	GPIO.26	26	12
13	23	GPIO.23	33 || 34	0v		
19	24	GPIO.24	35 || 36	GPIO.27	27	16
26	25	GPIO.25	37 || 38	GPIO.28	28	20
		0v	39 || 40	GPIO.29	29	21
BCM	wPi	Name	Physical	Name	wPi	BCM

Por exemplo, o pino físico 5 é o pino 3 segundo a numeração de Raspberry, que por sua vez é o pino 9 segundo a numeração de WiringPi.

Escolhi usar os pinos wiringPi (0, 1) para motor A e (2, 3) para motor B. Isto é, os pinos físicos (11, 12) e (13, 15). Você pode escolher outros. Ligue Raspberry, rode (por exemplo) o seguinte programa e monitore as saídas dos pinos 0, 1, 2 e 3 com um multímetro, osciloscópio ou LEDs em série com resistores de 270Ω. Verifique que as saídas GPIO estão ligando (3,3V) e desligando (0V) a cada 2 segundos. Compile com opção -w para linkar com biblioteca wiringPi.

```
//blink2.cpp
//compila blink2 -w
#include <wiringPi.h>
int main () {
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  pinMode (1, OUTPUT) ;
  pinMode (2, OUTPUT) ;
  pinMode (3, OUTPUT) ;
  for (int i=0; i<20; i++) {
    digitalWrite (0, HIGH) ;
    digitalWrite (1, HIGH) ;
    digitalWrite (2, HIGH) ;
    digitalWrite (3, HIGH) ;
    delay (2000) ;
    digitalWrite (0,  LOW) ;
    digitalWrite (1,  LOW) ;
    digitalWrite (2,  LOW) ;
    digitalWrite (3,  LOW) ;
    delay (2000) ;
  }
}
```

Agora, vamos testar softPwm. Rode (por exemplo) o seguinte programa e monitore as saídas dos pinos 0, 1, 2 e 3 (pinos físicos 11, 12, 13 e 15) com um multímetro ou osciloscópio. No multímetro, deve observar tensões 0,7x3,3V=1,65V e 0,3x3,3V=0,99V. No osciloscópio deve observar uma onda retangular de 100Hz com duty cycle de 70% e 30%.

```
//pwm1.cpp
//compila pwm1 -c -w
#include <cekeikon.h>
#include <wiringPi.h>
#include <softPwm.h>
int main () {
  wiringPiSetup () ;
  if (softPwmCreate(0, 0, 100)) erro("erro");
  if (softPwmCreate(1, 0, 100)) erro("erro");
  if (softPwmCreate(2, 0, 100)) erro("erro");
  if (softPwmCreate(3, 0, 100)) erro("erro");
  for (int i=0; i<20; i++) {
    softPwmWrite(0, 70);
    softPwmWrite(1, 70);
    softPwmWrite(2, 70);
    softPwmWrite(3, 70);
    delay (2000) ;
    softPwmWrite(0, 30);
    softPwmWrite(1, 30);
    softPwmWrite(2, 30);
    softPwmWrite(3, 30);
    delay (2000) ;
  }
}
```

Depois de certificar que a biblioteca WiringPi está funcionando, vamos fazer as ligações (figuras 4, 5 e 6). Faça todas as ligações com Raspberry e Ponte-H desenergizados. Ligue primeiro as saídas da ponte-H aos dois motores (conectores 1 e 2 no motor A e 13 e 14 no motor B, figura 6).

Depois, conecte os 4 fios GPIO da Raspberry às 4 entradas do Ponte-H (conectores 8, 9, 10 e 11 na figura 6). Além disso, você deve ligar entre si as terras da Raspberry (0V) e da ponte-H (pino 5). Isto é necessário para quando Raspberry e Ponte-H utilizarem fontes de alimentação independentes.

Após certificar-se de que não há problemas nas ligações, ligue a alimentação da ponte-H (5V, 1A do powerbank) e Raspberry (fonte de alimentação 5V, 2,5A ou saída 5V, 2A do powerbank). Teste a rotação dos motores executando o programa abaixo. Os motores esquerdo e direito devem rodar para frente e para trás com meia velocidade (PWM 60%).

Não iremos usar nunhuma entrada GPIO da Raspberry, mas é bom saber que o autor de WiringPi avisa: "Remember: The Raspberry Pi is a 3.3 volt device! Attempting to directly connect to any 5V logic system will very likely result in tears…"

```
//pwmroda2.cpp
//compila pwmroda2 -c -w
#include <cekeikon.h>
#include <wiringPi.h>
#include <softPwm.h>
int main() {
  wiringPiSetup();
  if (softPwmCreate(0, 0, 100)) erro("erro");
  if (softPwmCreate(1, 0, 100)) erro("erro");
  if (softPwmCreate(2, 0, 100)) erro("erro");
  if (softPwmCreate(3, 0, 100)) erro("erro");
  for (int i=0; i<2; i++) {
    softPwmWrite(0, 60);
    softPwmWrite(1, 0);
    delay(2000);
    softPwmWrite(0, 0);
    softPwmWrite(1, 60);
    delay(2000);
    softPwmWrite(0, 0);
    softPwmWrite(1, 0);
    delay(2000);

    softPwmWrite(2, 60);
    softPwmWrite(3, 0);
    delay(2000);
    softPwmWrite(2, 0);
    softPwmWrite(3, 60);
    delay(2000);
    softPwmWrite(2, 0);
    softPwmWrite(3, 0);
    delay(2000);
  }
}
```