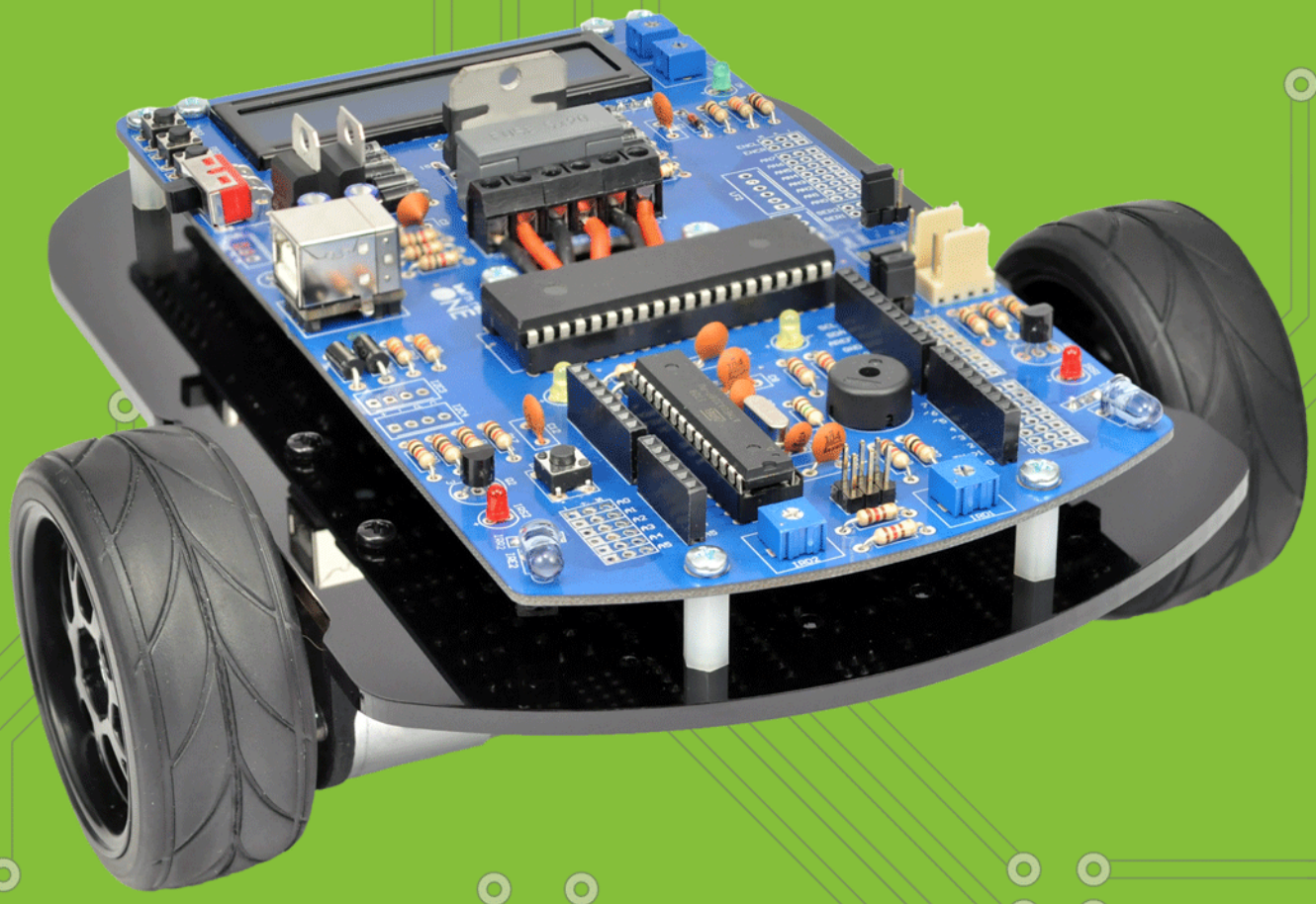
bot'n roll ONE

build your own robot



manual de software

www.botnroll.com

CONTEÚDO

Revisão do Documento: 15 de Fevereiro de 2016

1. INTRODUÇÃO

O Bot'n Roll ONE A é programado usando linguagem C com o ambiente de programação Arduino IDE. O microcontrolador ATmega328 presente no robô possui o *bootloader* do Arduino Uno, logo o robô é programado como se de um Arduino Uno se tratasse.

O robô possui um segundo microcontrolador, um PIC18F45K22 fornecido pré-programado com *software* desenvolvido pela botnroll.com. No Bot'n Roll ONE A funciona como um dispositivo escravo "***slave***" que executa as ordens de comando do "***master***" ATmega328.

Os dois microcontroladores do Bot'n Roll ONE A comunicam entre si através do barramento SPI "***Serial Peripheral Interface***". Os microcontroladores trocam informação de uma forma coordenada e bem definida. Para isso foi desenvolvido um protocolo de transferência de dados entre o master e o *slave*. O master utiliza uma lista de comandos que correspondem a ordens de controlo e cada comando gera uma resposta por parte do *slave*. A listagem de comandos e a forma como os dados são transmitidos entre *master* e *slave* estão definidos na biblioteca BnrOneA.

A biblioteca para Arduino BnrOneA torna possível ao utilizador controlar o robô de uma forma simples e para isso basta que use corretamente os comandos da biblioteca no Arduino IDE. Estes comandos estão listados e explicados neste manual.

Embora os dois microcontroladores possam ser programados em linguagem C pelo utilizador, somente o ATmega328 com *bootloader* Arduino é programado no dia-a-dia com recurso à biblioteca BnrOneA.

O PIC18F45K22 pode ser programado em linguagem C usando o ambiente de programação MPLABX IDE e o compilador XC8 da *Microchip* ou outro *software* compatível. No entanto, isto deve ser feito somente por utilizadores avançados pois programar o PIC18F45K22 para incluir uma nova funcionalidade requer que se atualize também a biblioteca BnrOneA para que o Arduino consiga utilizar a nova funcionalidade. Contacta a botnroll.com se gostarias de ver uma nova funcionalidade implementada no teu Bot'n Roll ONE A!

1.1 PROGRAMAR O BOT'N ROLL ONE A

Para programar o Bot'n Roll ONE A é necessário que tenhas o teu computador preparado com todas as ferramentas necessárias, ou seja:

- VCP driver instalado, o driver para porta USB do Bot'n Roll ONE A (ver ANEXO A);
- Arduino IDE instalado (ver ANEXO B);
- Biblioteca BnrOneA instalada no Arduino IDE (ver ANEXO B).

Para informações detalhadas sobre a instalação dos itens acima referidos consulta os anexos A e B no final deste manual.

A linguagem C é também uma ferramenta necessária para a programação do Bot'n Roll ONE A. Se ainda não estás muito à vontade com a linguagem C, tens os exemplos da biblioteca que são um bom guia para te iniciares neste mundo da programação. Consulta também as apresentações da RoboParty sobre programação em C, e claro, na internet existem milhares de páginas que explicam a linguagem C.

1.1.1 Arduino IDE

O ambiente de desenvolvimento Arduino contém um editor de texto para escrever o código, uma área de mensagens, uma consola de texto, uma barra de ferramentas com as funções mais importantes e ainda uma série de menus. Efetua a ligação ao *hardware* Arduino do Bot'n Roll ONE A para transferir o código e comunicar com o robô.

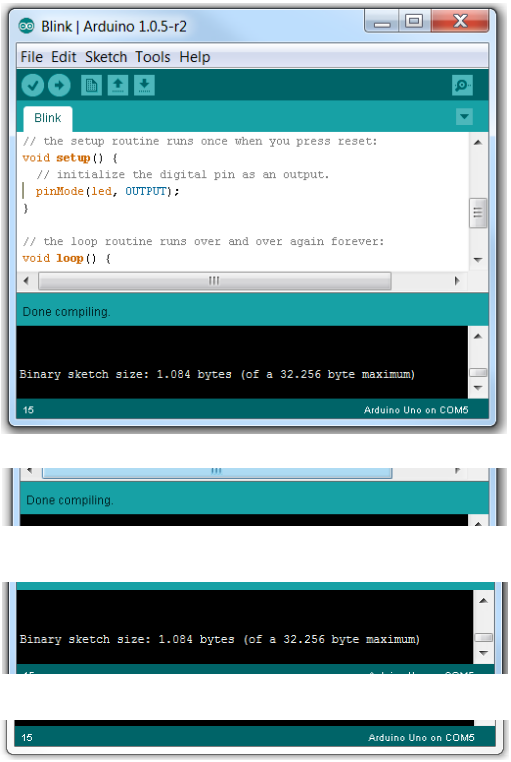


Fig. 1: Módulos do Arduino IDE

Um programa para Arduino tem o nome de "***sketch***", é escrito no editor de texto e guardado com a extensão ".ino" no teu computador.

A área de mensagens apresenta informação sobre a gravação e exportação dos programas e também apresenta os erros.

A consola apresenta mensagens de texto com informação detalhada sobre os erros e outra informação.

No canto inferior direito da janela é apresentada informação sobre a placa a ser programada e a porta série em utilização.

Os botões na barra de ferramentas e as suas funções:

Verify: Verificar a existência de erros no código.

Upload: Compilar o código e enviar para o Arduino.

New: Criar um novo *sketch*.

Open: Abrir um *sketch* guardado no computador.

Save: Guardar o ***sketch***.

Serial Monitor: Abrir a monitorização da porta série.

O *serial monitor* permite visualizar dados enviados do Arduino para o computador e também permite o envio de dados do computador para o Arduino. É muito útil na programação pois consegues imprimir aqui texto e o valor das variáveis e assim efetuar o "***debug***" (procura de erros) do teu programa. Quando abres o *serial monitor*, o teu programa no Arduino reinicia.

1.1.2 Biblioteca BnrOneA para Arduino

Uma biblioteca é um conjunto código "pré-fabricado" que podes inserir e utilizar no teu programa. Para usares a biblioteca **BnrOneA** basta que a incluas no teu código:

#include<BnrOneA.h>

E que cries uma instância para a classe BnrOneA:

BnrOneA one;

A partir daqui tens acesso a todas as funções da biblioteca que são precedidas pela instância que definiste ou seja: **one.***função_da_biblioteca()*;

Uma biblioteca é normalmente criada para manipulação de dados ou de *hardware* e tem sempre dois ficheiros pelo menos, mas no caso do Arduino há ainda um ficheiro adicional com a extensão "**.txt"**.

- Um ficheiro com a extensão "**.h**" ("***header***") que contém a listagem de todas as funções, comandos e definições da biblioteca;
- Um ficheiro com a extensão "**.cpp**" ("***c++ source***") com a codificação de todas as funções apresentadas no ficheiro *header*.
- Um ficheiro ***keywords.txt*** que permite ao Arduino IDE identificar as funções da biblioteca e apresentá-las com uma coloração diferente do resto do código.

A biblioteca BnrOneA foi criada para a manipulação do *hardware* associado ao PIC18F45K22 e permite ao Arduino interagir com ele através do barramento de comunicação SPI. O Arduino tem acesso a todo o *hardware* e funcionalidades definidas na biblioteca e no *software* do PIC18F45K22. A biblioteca BnrOneA e o *software* do PIC18F45K22 foram feitos "um para o outro" e qualquer alteração num deles requer que se ajuste o outro também.

Todas as funções da biblioteca BnrOneA estão especificadas e explicadas no ponto 2 deste manual.

1.1.3 Linguagem de Programação C

A linguagem C foi desenvolvida em 1972 por Dennis Ritchie nos laboratórios Bell em Nova Jersey. Surgiu com o intuito de ser uma linguagem poderosa e rápida para ser utilizada no sistema operativo Unix que estava a desenvolver. Ao longo do tempo foi melhorada e atualizada mostrando-se muito robusta e fiável e passou a ser utilizada também por outros sistemas operativos como o Windows, MacOs e Linux. Está em constante evolução desde que surgiu a primeira versão conhecida como "**K&R C**". Em 1989 surgiu a primeira especificação como padrão pelo instituto norte-americano de padrões o "**ANSI C**". Em 1990 o "ISO C" pela Organização Internacional para a Padronização. Em 1999 surgiu o *standard* "**C99**" e a mais recente revisão data de Dezembro de 2011 a ISO**/IEC 9899:2011** mais conhecido como "**C11**".

Todas estas atualizações e revisões visam a utilização da linguagem C no desenvolvimento de programas para computadores pessoais onde os recursos de processamento e memória não são uma limitação. Para a utilização em microcontroladores, é utilizada uma versão "mais leve" da linguagem C pois os recursos de memória e processamento são limitados. Assim, para programares com sucesso o teu Bot'n Roll ONE A necessitas de saber apenas algumas regras básicas da linguagem para Arduino e o funcionamento de alguns comandos.

Todos os programas para Arduino têm duas rotinas, ou funções, que são obrigatórias. A rotina "***setup()"***, configuração, é executada somente uma vez no arranque do teu programa. Aqui deve ser colocado todo o código necessário para inicializar variáveis, configurar pinos de entrada e saída, configurar comunicação SPI, Série, I2C, enfim, todas as configurações necessárias.

Depois da configuração, o teu programa entra na rotina ***loop()*** e lá permanece indefinidamente. O termo ***loop*** significa **ciclo** e neste caso é um ciclo infinito pois quando o programa atinge o final do ciclo, volta ao início e começa tudo de novo! É aqui que escreves o programa e crias a inteligência para o teu robô!

A programação em C propriamente dita, não a explicamos neste manual, remetemos para os exemplos da biblioteca BnrOneA e do Arduino em geral. Todo o código está devidamente comentado e terás que experimentar e testar para perceberes como funciona. Deixamos-te no entanto algumas dicas que aprendemos ao longo do tempo na **botnroll.com**:

- Cria programas novos a partir dos exemplos básicos. Experimenta juntar 3 ou 4 funcionalidades do robô no mesmo programa a partir dos exemplos básicos!
  - Um programa raramente funciona à primeira! Não desanimes, procura o problema e resolve-o!
- Insere código aos poucos e vai testando para ver se tudo acontece como esperado.
- Para um programa ficar bom vais perder mais tempo a testá-lo que a escrevê-lo!
  - Usa as ferramentas de *debug* como o LED, o serial monitor ou o LCD para imprimir o valor das variáveis e verificar se o programa passa numa determinada zona do código.
- Programar é como praticar um desporto novo, no início é doloroso pois não tens a condição física necessária, não sabes as regras e andas um pouco perdido. Nos treinos praticas, aprendes e melhoras em todos os aspetos. Com o resultado do teu trabalho acabas por fazer parte da equipa principal!

O Bot'n Roll ONE A permite a interação com um leque de *hardware* muito vasto. Existem extras, vulgarmente chamados "*shields*" para Arduino para fazer praticamente tudo o que imaginas e são compatíveis com o Bot'n Roll ONE A! Todos os *shields* possuem bibliotecas para te ajudar na sua utilização e integração e a tua imaginação é o limite!

2. FUNÇÕES DA BIBLIOTECA BNRONEA PARA ARDUINO

A biblioteca BnrOneA possui funções, ou rotinas, que permitem ao Arduino (ATmega328) aceder a todos os periféricos controlados pelo PIC18F45K22. Estas rotinas foram divididas em três grupos: configuração, leitura e escrita. Muitas destas funções recorrem ao uso de **parâmetros/argumentos** para troca de informação, ou seja, o envio e/ou receção de valores de variáveis.

Um argumento é uma qualquer expressão dentro dos parêntesis de uma função por exemplo:

- **one.spiConnect(sspin);**

O **argumento** é **sspin** e é transferido como **parâmetro** da função **spiConnect** durante a execução da função.

Uma função também pode **devolver** um valor como resultado da sua execução:

- **float battery = one.readBattery();**

A função de leitura da bateria devolve como resultado da sua execução o valor da bateria, o qual é armazenado na variável *battery*.

Apresenta-se de seguida uma listagem de todas as funções da biblioteca BnrOneA extraídas do ficheiro BnrOneA.h.

Listagem das funções da biblioteca BnrOneA

Setup routines	**Reading routines**
void spiConnect(byte sspin);	byte obstacleSensors();
void minBat(float batmin);	byte readIRSensors();
void obstacleEmitters(boolean state);	IntreadAdc(byte);
	int readAdc0();
	int readAdc1();
	int readAdc2();
Writing routines	int readAdc3();
void servo1(byte position);	int readAdc4();
void servo2(byte position);	int readAdc5();
void led(boolean state);	int readAdc6();
void move(intspeedL,intspeedR);	int readAdc7();
void stop();	intreadButton();
void brake(byte torqueL,bytetorqueR);	float readBattery();
void resetEncL();	intreadEncL();
void resetEncR();	intreadEncR();
	intreadEncLInc();
	intreadEncRInc();
LCD Line 1 write routines	**LCD Line 2 write routines**
void lcd1(byte string[]);	void lcd2(byte string[]);
void lcd1(const char string[]);	void lcd2(const char string[]);
void lcd1(int number);	void lcd2(int number);
void lcd1(unsigned int number);	void lcd2(unsigned int number);
void lcd1(long int number);	void lcd2(long int number);
void lcd1(double number);	void lcd2(double number);
void lcd1(const char string[],int number);	void lcd2(const char string[],int number);
void lcd1(const char string[],unsigned int number);	void lcd2(const char string[],unsigned int number);
void lcd1(const char string[],long int number);	void lcd2(const char string[],long int number);
void lcd1(const char string[],double number);	void lcd2(const char string[],double number);
void lcd1(int num1, int num2);	void lcd2(int num1, int num2);
void lcd1(unsigned int num1, unsigned int num2);	void lcd2(unsigned int num1, unsigned int num2);
void lcd1(int num1, int num2, int num3);	void lcd2(int num1, int num2, int num3);
void lcd1(int num1, int num2, int num3, int num4);	void lcd2(int num1, int num2, int num3, int num4);
void lcd1(unsigned int num1, unsigned int num2, unsigned int num3);	void lcd2(unsigned int num1, unsigned int num2, unsigned int num3);

void lcd1(unsigned int num1, unsigned int num2, unsigned int num3, unsigned int num4);	void lcd2(unsigned int num1, unsigned int num2, unsigned int num3, unsigned int num4);

2.1.1 spiConnect(sspin)

Descrição:

Inicializa o barramento de comunicação SPI configurando os pinos SCK, MOSI e SS como saídas. Coloca os pinos SCK e MOSI no estado baixo (0V) e SS no estado alto (5V). No Bot'n Roll ONE A, o pino SS "*Slave Select*" para a comunicação SPI entre o ATmega328 e o PIC18F45K22,corresponde por defeito à saída digital 2 mas pode ser alterado removendo o *jumper* SSP e efetuando a ligação desejada. O pino SS nativo do ATmega328 pode ser utilizado para a comunicação SPI com um qualquer *shield* que utilize esta ligação.

Parâmetros:

sspin: a saída digital a ser usada como "*Slave Select*" na comunicação SPI entre o ATmega328 e o PIC18F45K22. (*byte*)

Devolve:

Nada

Exemplo:

```
#include <BnrOneA.h>          // Bot'n Roll ONE A library
#include <SPI.h>              // SPI communication library required by BnrOne.cpp
BnrOneA one;                  // declaration of object variable to control the Bot'n Roll ONE A
#define SSPIN 2               // Slave Select (SS) pin for SPI communication
void setup()
{
    one.spiConnect(SSPIN);    // start the SPI communication module
}
```

2.1.2 minBat(batmin)

Descrição:

Define o valor mínimo de tensão da bateria a partir do qual o robô bloqueia o movimento e a escrita na linha 2 do LCD. Este valor de configuração é escrito na EEPROM do PIC18F45K22 e não se perde ao desligar o Bot'n Roll ONE A. Durante a escrita na EEPROM o PIC18F45K22 fica incomunicável, logo não deve ser enviado qualquer comando durante este processo que tem a duração de aproximadamente 5ms. Sempre que a tensão da bateria baixar do valor mínimo configurado o robô não responde a comandos de movimento e apresenta uma mensagem de bateria baixa na linha 2 do LCD. Esta função ajuda a preservar a vida das baterias e é muito útil no caso da utilização de baterias de lítio que se destroem se a tensão baixar do valor mínimo de segurança.

Parâmetros:

batmin: O valor mínimo da tensão da bateria (float)

Devolve:

Nada

Exemplo:

```
#include <BnrOneA.h>          // Bot'n Roll ONE A library
#include <SPI.h>              // SPI communication library required by BnrOne.cpp
BnrOneA one;                  // declaration of object variable to control the Bot'n Roll ONE A
#define SSPIN 2               // Slave Select (SS) pin for SPI communication
void setup()
{
    one.spiConnect(SSPIN);    // start the SPI communication module
    one.minBat(8.5);          // define de minimum battery voltage
    delay(5);                           // wait 5ms
}
```

2.1.3 obstacleEmitters(state)

Descrição:

Controla o estado dos LED's emissores de infravermelhos. Os emissores de infravermelhos podem ser desligados se necessário e a normal deteção de obstáculos fica desativada. O valor 0 desliga os emissores e o valor 1 liga os emissores.

Parâmetros:

state: o estado a colocar os LED's (*boolean*)

Devolve:

Nada

Exemplo:

```
#include <BnrOneA.h>              // Bot'n Roll ONE A library
#include <SPI.h>                  // SPI communication library required by BnrOne.cpp
BnrOneA one;                      // declaration of object variable to control the Bot'n Roll ONE A
#define SSPIN 2                   // Slave Select (SS) pin for SPI communication
void setup()
{
    one.spiConnect(SSPIN);        // start the SPI communication module
    one.obstacleEmitters(ON);     // activate IR emitter LEDs
}
```

2.2 FUNÇÕES DE LEITURA

2.2.1 obstacleSensors()

Descrição:

Devolve o resultado da leitura dos obstáculos e existem quatro situações possíveis:

- **0**: não há obstáculos
- **1**: obstáculo detetado no sensor esquerdo
- **2**: obstáculo detetado no sensor direito
- **3**: obstáculo detetado em ambos os sensores

A leitura de obstáculos é efetuada a cada 20ms pelo PIC18F45K22 e esta função devolve o resultado da última leitura. Para se efetuar a leitura de obstáculos é necessário que os emissores de infravermelhos estejam ativados!

Parâmetros:

Nenhum

Devolve:

O valor da leitura dos obstáculos (*byte*)

Exemplo:

```
...
void loop()
{
    byte obstacle=one.obstacleSensors(); // read obstacle sensors
...
}
```

2.2.2 readIRSensors()

Descrição:

Devolve o estado atual dos sensores de infravermelhos SHARP GP1UX511QS e existem quatro situações possíveis:

- **0**: ambos os sensores no estado alto "*High*"
- **1**: sensor esquerdo a alto "*High*" e sensor direito a baixo "*Low*"
- **2**:sensor esquerdo a baixo "*Low*" e sensor direito a alto "*High*"
- **3**:ambos os sensores no estado baixo "*Low*"

Esta função força o PIC18F45K22 a efetuar uma leitura instantânea do estado dos sensores de infravermelhos. Nota que os sensores SHARP GP1UX511QS apresentam um sinal invertido em relação ao do emissor de infravermelhos.

Parâmetros:

Nenhum

Devolve:

O estado atual dos sensores de infravermelhos (*byte)*

Exemplo:

```
...
void loop()
{
    byte obstacle=one.readIRSensors(); // read actual IR sensors state
...
}
```


2.2.3 readAdc(byte)

Descrição:

Devolve o valor da conversão ADC "*Analog to Digital Conversion*" associada ao PIC18F45K22 para o canal especificado no parâmetro da função. O parâmetro define qual dos canais analógicos AN0 a AN7 do Bot'n Roll ONE A se deseja a conversão e admite os seguintes valores:

- **0**(corresponde ao canal AN0)
- **1** (corresponde ao canal AN1)
- **2** (corresponde ao canal AN2)
- **3** (corresponde ao canal AN3)
- **4** (corresponde ao canal AN4)
- **5** (corresponde ao canal AN5)
- **6** (corresponde ao canal AN6)
- **7** (corresponde ao canal AN7)

O valor obtido na conversão varia entre 0 e 1023 pois o PIC18F45K22 possui um conversor ADC de 10bits.

Parâmetros:

O canal ADC (*byte*)

Devolve:

O valor da conversão ADC (int)

Exemplo:

```
...
void loop()
{
    for(i=0;i<8;i++)
    {
    adc[i]=one.readAdc(i);        // read the ADC conversion value
    ...
    }
}
```


2.2.4 readAdcX()

Descrição:

Devolve o valor da conversão ADC "*Analog to Digital Conversion*" associada ao PIC18F45K22 do canal ANX do Bot'n Roll ONE A.O valor obtido na conversão varia entre 0 e 1023 pois o PIC18F45K22 possui um conversor ADC de 10bits.

A letra "X" representa o canal ADC que se pretende ler e deverá ser substituído por um número de 0 a 7.

Parâmetros:

Nenhum

Devolve:

O valor da conversão ADC (int)

Exemplo:

```
...
void loop()
{
    int adc1 =one.readAdc1();    // read the ADC conversion value
    int adc6 =one.readAdc6();    // read the ADC conversion value
...
}
```


2.2.5 readButton()

Descrição:

Indica qual dos botões de pressão PB1, PB2 ou PB3 está a ser pressionado. A função devolve um dos possíveis valores como resultado:

- **0**: nenhum botão está pressionado
- **1**: PB1 pressionado
- **2**: PB2 pressionado
- **3**: PB3 pressionado

Se mais que um botão for pressionado simultaneamente é devolvido o valor mais baixo.

Parâmetros:

Nenhum

Devolve:

Botão pressionado (*int*)

Exemplo:

```
...
void loop()
{
    int pbutton=one.readButton();     // read the Push Button value
...
}
```

2.2.6 readBattery()

Descrição:

Leitura do valor da tensão da bateria em Volts.

Parâmetros:

Nenhum

Devolve:

Valor da tensão da bateria (float)

Exemplo:

```
...
void loop()
{
    float battery=one.readBattery();          // read battery voltage
...
}
```


2.2.7 readEncL()

Descrição:

Devolve a leitura da contagem do encoder esquerdo e efetua um *reset* ao encoder, limpando o valor da contagem. Esta função é útil para medir e controlar a velocidade de uma roda. Se a velocidade de uma roda for constante, para intervalos de tempo bem definidos deveremos obter valores de contagem iguais.

O encoder corresponde a um contador de 16-bit e a contagem varia entre -32768 e 32767.

Parâmetros:

Nenhum

Devolve:

Contagem do encoder esquerdo (*int*)

Exemplo:

```
...
void loop()
{
    int encL=one.readEncL();
...
}
```

2.2.8 readEncR()

Descrição:

Devolve a leitura da contagem do encoder direito e efetua um *reset* ao encoder, limpando o valor da contagem. Esta função é útil para medir e controlar a velocidade de uma roda. Se desejamos que a velocidade de uma roda seja constante, para intervalos de tempo bem definidos deveremos obter valores de contagem iguais.

O encoder corresponde a um contador de 16-bit e a contagem varia entre -32768 e 32767.

Parâmetros:

Nenhum

Devolve:

Contagem do encoder direito (*int*)

Exemplo:

```
...
void loop()
{
    int encR=one.readEncR();
...
}
```


2.2.9 readEncLInc()

Descrição:

Devolve a leitura da contagem incremental do encoder esquerdo. Esta função é útil para controlar a distância percorrida por uma roda. Sabendo que o incremento de uma unidade do valor do encoder corresponde a uma certa distância, controlamos a distância percorrida por uma roda. Não é efetuado qualquer *reset* ao valor do encoder. O encoder corresponde a um contador de 16-bit e a contagem varia entre -32768 e 32767. O utilizador deverá controlar a situação de *overflow* do encoder.

Parâmetros:

Nenhum

Devolve:

Contagem incremental do encoder esquerdo (*int*)

Exemplo:

```
…
void loop()
{
    int encL=one.readEncLInc();
…
}
```

2.2.10 readEncRInc()

Descrição:

Devolve a leitura da contagem incremental do encoder direito. Esta função é útil para controlar a distância percorrida por uma roda. Sabendo que o incremento de uma unidade do valor do encoder corresponde a uma certa distância, controlamos a distância percorrida por uma roda. Não é efetuado qualquer *reset* ao valor do encoder. O encoder corresponde a um contador de 16-bit e a contagem varia entre -32768 e 32767. O utilizador deverá controlar a situação de *overflow* do encoder.

Parâmetros:

Nenhum

Devolve:

Contagem incremental do encoder direito (*int*)

Exemplo:

```
…
void loop()
{
    int encR=one.readEncRInc();
…
}
```


2.3 FUNÇÕES DE COMANDO

2.3.1 servo1(position)

Descrição:

Movimenta o servo ligado em SER1. O parâmetro da função define a posição angular que varia de 0 a 180.

Um servo normal posiciona-se no ângulo definido pelo parâmetro da função e o ângulo varia de 0º a 180º.

Um servo de rotação contínua varia a velocidade de rotação de acordo com o valor do parâmetro da função. '0' corresponde à rotação máxima num determinado sentido, 180 à rotação máxima no sentido inverso e 90 corresponde ao servo parado.

Parâmetros:

position: a posição angular do servo (*byte*)

Devolve:

Nada

Exemplo:

```
...
void loop()
{
    one.servo1(70);
...
}
```

2.3.2 servo2(position)

Descrição:

Movimenta o servo ligado em SER2. O parâmetro da função define a posição angular que varia de 0 a 180.

Um servo normal posiciona-se no ângulo definido pelo parâmetro da função e o ângulo varia de 0° a 180°.

Um servo de rotação contínua varia a velocidade de rotação de acordo com valor do parâmetro da função. 0 corresponde à rotação máxima num determinado sentido, 180 à rotação máxima no sentido inverso e 90 corresponde ao servo parado.

Parâmetros:

position: a posição angular do servo (*byte*)

Devolve:

Nada

Exemplo:

```
...
void loop()
{
      one.servo2(130);
...
}
```

2.3.3 led(state)

Descrição:

Acende ou apaga o LED do Bot'n Roll ONE A. O estado do LED é definido pelo parâmetro da função e permite dois estados:

- **0**: LED desligado
- **1**: LED ligado

Parâmetros:

state: o estado do LED (*boolean*)

Devolve:

Nada

Exemplo:

```
...
void loop()
{
    one.led(HIGH);          // turn LED ON
...
}
```

2.3.4 move(speedL,speedR)

Descrição:

Movimenta os motores do Bot'n Roll ONE A. Os parâmetros definem a velocidade de cada motor, esquerdo e direito, que varia de -100 a 100. O valor -100 corresponde à velocidade máxima no sentido inverso, 100 corresponde à velocidade máxima no sentido direto e 0 corresponde à paragem do motor.

Parâmetros:

speedL: velocidade do motor esquerdo (*int*)

speedR: velocidade do motor direito (int)

Devolve:

Nada

Exemplo:

```
...
void loop()
{
    one.move (70,70);          // Move forward at speed 70.
...
}
```


2.3.5 stop()

Descrição:

Paragem dos motores do Bot'n Roll ONE A. Corta a energia aos motores e estes rodam livremente até pararem, não é aplicado qualquer binário de travagem.

Parâmetros:

Nenhum

Devolve:

Nada

Exemplo:

```
...
void loop()
{
    one.stop ();          // Stop motors without braking torque
...
}
```

2.3.6 brake(torqueL,torqueR)

Descrição:

Paragem dos motores do Bot'n Roll ONE A aplicando binário de travagem. A potência de travagem de cada motor é definida pelos parâmetros da função e varia entre 0 e 100. O valor 0 corresponde a paragem sem binário de travagem. O valor 100 corresponde a paragem com binário de travagem máximo e instantaneamente os motores bloqueiam!

Parâmetros:

torqueL: binário de travagem do motor esquerdo (*byte*)

torqueR: binário de travagem do motor direito (*byte*)

Devolve:

Nada

Exemplo:

```
...
void loop()
{
    one.brake (60,60);          // Brake motors with 60% brake power.
...
}
```


2.3.7 resetEncL()

Descrição:

Limpa a contagem do encoder esquerdo ficando o seu valor igual a 0.

Parâmetros:

Nenhum

Devolve:

Nada

Exemplo:

```
...
void loop()
{
    one.resetEncL();          // Reset left encoder
...
}
```

2.3.8 resetEncR()

Descrição:

Limpa a contagem do encoder direito ficando o seu valor igual a 0.

Parâmetros:

Nenhum

Devolve:

Nada

Exemplo:

```
...
void loop()
{
    one.resetEncR();          // Reset right encoder
...
}
```


2.4 FUNÇÕES DE ESCRITA NO LCD

2.4.1 lcdX(string[])

Descrição:

Imprime na linha X do LCD um *array* de caracteres (*string*) ou texto entre aspas ("texto a enviar") enviados como parâmetro. Como parâmetros são aceites os tipo de variável (*char*) e (*constchar*).O tamanho máximo em caracteres para a *string* ou texto entre aspas é de 17 que corresponde a 16 caracteres escritos no LCD mais o caractere de terminação '**\0**'.

A letra "X" representa a linha do LCD que se pretende escrever e deverá ser substituído por 1 ou 2.

Parâmetros:

string[]: *array* com os caracteres a serem escritos no LCD (*char*) ou (*constchar*).

Devolve:

Nada

Exemplo:

```
...
char string[]=" String Test ";       // declare and initialize the string
void loop()
{
    one.lcd1(string);                //print string on LCD line 1
    one.lcd2(" Text to LCD! ");          //print text on LCD line 2
...
}
```

2.4.2 lcdX(number)

Descrição:

Imprime na linha X do LCD um número ou variável que é enviado(a) como parâmetro. O número pode ser uma variável dos tipos (*int*), (*unsigned int*), (*long int*), (*double*) ou (float).

A letra "X" representa a linha do LCD que se pretende escrever e deverá ser substituído por 1 ou 2.

Parâmetros:

number: número ou variável a imprimir (*int*), (*unsigned int*), (*long int*), (*double*) ou (*float*).

Devolve:

Nada

Exemplo:

```
...
intvar1 =-32768;                // declare and initialize the variable
unsigned int var2 = 0;          // declare and initialize the variable
long int var3 = - 2147483648;   // declare and initialize the variable
float var4=123.12;              // declare and initialize the variable
double var5=123.12;             // declare and initialize the variable

void loop()
{
    one.lcd1(var1);             //print variable value on LCD
    ...
    one.lcd2(var2);             //print variable value on LCD
    ...
    one.lcd1(var3);             //print variable value on LCD
    ...
    one.lcd2(var4);             //print variable value on LCD
    ...
    one.lcd1(var5);             //print variable value on LCD
    ...
    one.lcd2(32767);            //print a number on LCD
    ...
    one.lcd1(2147483647);       //print a big number on LCD
    ...
    one.lcd2(321.01);                   //print single precision number on LCD
...
}
```


2.4.3 lcdX(string[],number)

Descrição:

Imprime na linha X do LCD o texto e o número enviados como parâmetros. Como texto é aceite o tipo de variável (*constchar*), ou seja, texto entre aspas. O número total de caracteres a imprimir não deverá ser maior que 16.O número pode ser uma variável dos tipos (*int*), (*unsigned int*), (*long int*), (*double*) ou (*float*).

A letra "X" representa a linha do LCD que se pretende escrever e deverá ser substituído por 1 ou 2.

Parâmetros:

string[]: caracteres a serem escritos no LCD (*constchar*).

number: número ou variável a imprimir (*int*), (*unsigned int*), (*long int*), (*double*) ou (*float*).

Devolve:

Nada

Exemplo:

```
...
intvar1 =-32768;                // declare and initialize the variable
unsigned int var2 = 0;          // declare and initialize the variable
long int var3 = - 2147483648;   // declare and initialize the variable
float var4=123.12;              // declare and initialize the variable
double var5=123.12;             // declare and initialize the variable

void loop()
{
    one.lcd1("Text: ", var1);               //print text and a variable value on LCD
    ...
    one.lcd2("Text: ", var2);               //print text and a variable value on LCD
    ...
    one.lcd1("Text: ",var3);                //print text and a variable value on LCD
    ...
    one.lcd2("Text: ",var4);                //print text and a variable value on LCD
    ...
    one.lcd1("Text: ",var5);                //print text and a variable value on LCD
    ...
    one.lcd2("Text: ",32767);               //print text and a number on LCD
    ...
    one.lcd1("Text: ",2147483647);          //print text and a big number on LCD
    ...
    one.lcd2("Text: ",321.01);              //print text and a single precision number on LCD
...
}
```


2.4.4 lcdX(num1, num2)

Descrição:

Imprime na linha X do LCD os números ou variáveis enviados (as) como parâmetros. Os números podem ser variáveis do tipo (*int*) ou (*unsigned int*).

A letra "X" representa a linha do LCD que se pretende escrever e deverá ser substituído por 1 ou 2.

Parâmetros:

num1: número ou variável a imprimir (*int*) ou (*unsigned int*).

num2: número ou variável a imprimir (*int*) ou (*unsigned int*).

Devolve:

Nada

Exemplo:

```
...
intvar1 =-32768;                // declare and initialize the variable
unsigned int var2 = 0;          // declare and initialize the variable

void loop()
{
    one.lcd1(var1 , 32767);             //print variable and number on LCD
    ...
    one.lcd2(var2, 65535);              //print variable and number on LCD
...
}
```


2.4.5 lcdX(num1, num2, num3)

Descrição:

Imprime na linha X do LCD os números ou variáveis enviados (as) como parâmetros. Os números podem ser variáveis do tipo (*int*) ou (*unsigned int*).

A letra "X" representa a linha do LCD que se pretende escrever e deverá ser substituído por 1 ou 2.

Parâmetros:

num1: número ou variável a imprimir (*int*) ou (*unsigned int*).

num2: número ou variável a imprimir (*int*) ou (*unsigned int*).

num3: número ou variável a imprimir (*int*) ou (*unsigned int*).

Devolve:

Nada

Exemplo:

```
...
intvar1=-32768;                 // declare and initialize the variable
unsigned int var2 = 0;          // declare and initialize the variable

void loop()
{
        one.lcd1(var1 ,32767, var2);          //print variable value and number on LCD
...
}
```


2.4.6 lcdX(num1 , num2 , num3 , num4)

Descrição:

Imprime na linha X do LCD os números ou variáveis enviados (as) como parâmetros. Os números podem ser variáveis do tipo (*int*) ou (*unsigned int*).

A letra "X" representa a linha do LCD que se pretende escrever e deverá ser substituído por 1 ou 2.

Parâmetros:

num1: número ou variável a imprimir (*int*) ou (*unsigned int*).

num2: número ou variável a imprimir (*int*) ou (*unsigned int*).

num3: número ou variável a imprimir (*int*) ou (*unsigned int*).

num4: número ou variável a imprimir (*int*) ou (*unsigned int*).

Devolve:

Nada

Exemplo:

```
...
intvar1 =-32768;                    // declare and initialize the variable
unsigned int var2 = 0;              // declare and initialize the variable

void loop()
{
    one.lcd2(var1 ,32767, var2 , 65535);    //print variable value and number on LCD
...
}
```


ANEXO A: INSTALAÇÃO DO VCP DRIVER DO CONVERSOR USB-SÉRIE (RS232)

O driver permite que o sistema operativo do teu computador comunique com o Bot'n Roll ONE A.

Para instalares o driver visita a página de suporte do Bot'nRoll ONE A http://botnroll.com/onea/ e faz *download* clicando em "**VCP Driver - Windows**"ou "**VCP Driver - Mac OS X**"de acordo com o teu sistema operativo. Assim que terminar o *download* descompacta o ficheiro com a extensão ".zip" eexecuta a aplicação.

Sempre que ligares o robô ao computador usando o cabo USB é criada uma porta COM virtual (VCP) pela qual é efetuada a comunicação entre o Bot'n Roll ONEA e o PC. A aplicação para a programação do robô usa esta porta para comunicar com o Bot'n Roll ONE A e desta forma transferir os programas para o robô.

O conversor USB-Série utilizado no Bot'n Roll ONE A é um **PoUSB12** da *PoLabs* usa o dispositivo ***Bridge* CP2102** da *Silicon Labs*.

ANEXO B:AMBIENTE DE PROGRAMAÇÃO ARDUINO

O *software* utilizado para a programação do robô é o Arduino IDE. Esta aplicação é necessária para fazer a edição dos programas em linguagem C. Serve também para transferir os teus programas para o Bot'n Roll ONEA.

B.1 INSTALAÇÃO DO ARDUINO IDE

Para a instalação do Arduino IDE, visita a página de suporte do Bot'n Roll ONE Ahttp://botnroll.com/onea/ e nos *downloads* essenciais clica em "**Arduino IDE - Windows"** ou "**Arduino IDE - Mac OS X"** de acordo com o teu sistema operativo.

Assim que o *download* terminar, descompacta o ficheiro com a extensão ".zip" e coloca a pasta extraída numa diretoria do teu computador a teu gosto.

Esta pasta contém várias subpastas e ficheiros, entre eles a aplicação "**arduino",** o executável que arranca o Arduino IDE. A subpasta "***libraries***" também é muito importante e contém todas as bibliotecas do ambiente Arduino. As bibliotecas são as tuas ferramentas de trabalho em programação.

B.2 INSTALAÇÃO DA BIBLIOTECA BNRONEA PARA ARDUINO

A biblioteca **BnrOneA** desenvolvida pela **botnroll.com** para o Arduino IDE possui todos os comandos necessários para o controlo do robô. Esta biblioteca deve ser instalada no Arduino IDE.

Na página de suporte do Bot'n Roll ONE A http://botnroll.com/onea/, faz o *download* do ficheiro **BnrOneA.zip** clicando em "**Biblioteca Arduino**".

Descompacta o ficheiro e coloca a pasta extraída "**BnrOneA**" dentro da subpasta "***libraries***" de que falamos no ponto anterior. Possuis agora todas as ferramentas necessárias para programares com sucesso o teu **Bot'n Roll ONE A!**

B.3 CONFIGURAÇÃO DA COMUNICAÇÃOCOM O ROBÔ

Antes de efetuares este passo, certifica-te que instalaste o VCP driver corretamente (ver ANEXO A). Conecta o Bot'n Roll ONEA ao computador usando o cabo USB fornecido. Neste momento, será atribuída automaticamente uma porta COM para a comunicação com o robô.

Abre o Arduino IDE e no separador "***Tools -> Board***" seleciona a placa "**Arduino Uno**". O **Bot´n Roll ONE A** será programado como se de um Arduino Uno se tratasse.

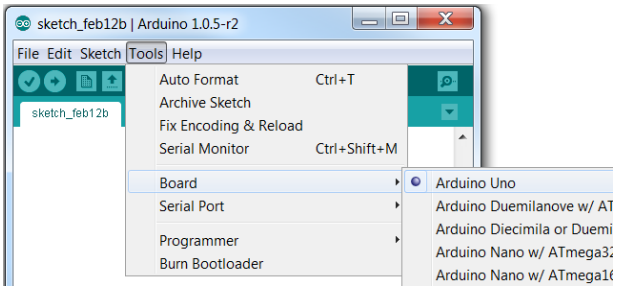


Fig. 2: Selecionar a placa a programar

No separador "***Tools -> Serial Port***" selecciona a porta COM atribuída ao **Bot'n Roll ONE A**.

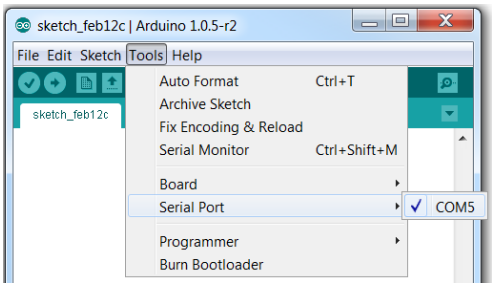


Fig. 3: Selecionar a Porta Série

Se nenhuma porta COM estiver disponível, o mais certo é não teres instalado corretamente o VCP driver do conversor USB-Série.

Abre o gestor de dispositivos do Windows e procura o item com a designação "Portas (COM e LPT)". Expandindo este item, verás todas as portas COM atribuídas.

"***Silicon Labs CP210x USB to UART Bridge***" é a designação que identifica a porta de ligação ao Bot'n Roll ONEA. (No exemplo da figura foi atribuída a porta **COM5**.)

Caso não apareça o item "***Silicon Labs CP210x USB to UART Bridge***" terás que instalar corretamente o VCP driver.

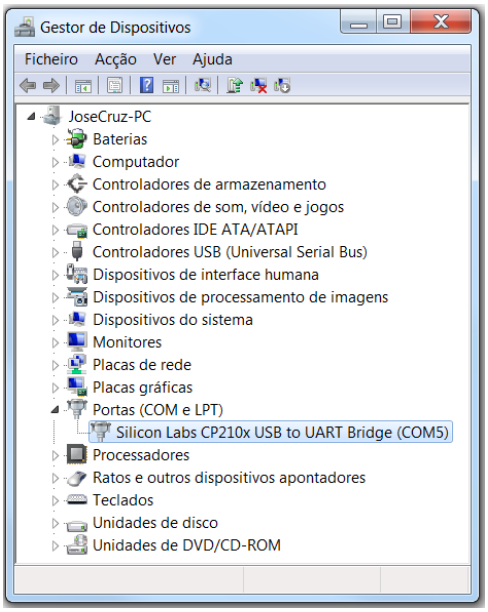


Fig. 4 Portas COM no Gestor de Dispositivos

B.4 CARREGAR UM PROGRAMA PARA O BOT'N ROLL ONE A

No ambiente de programação Arduino encontras vários programas de exemplo que podes carregar para o robô.

Clica em "***File -> Examples -> 01.Basics -> Blink***" e aparece uma nova janela com o código deste exemplo.

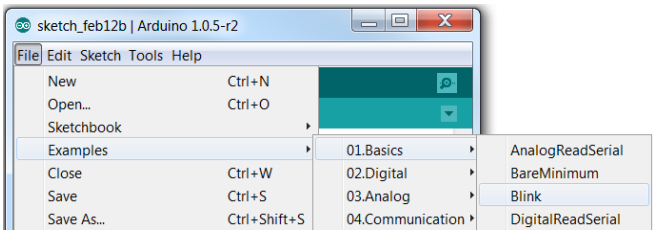


Fig. 5: Carregar um programa de exemplo

Clica em "***File -> Upload***" ou carrega no símbolo com a seta para o lado direito para enviar o programa para o robô. Assim que o *upload* terminar deverás ver o LED amarelo L a piscar a cada segundo!

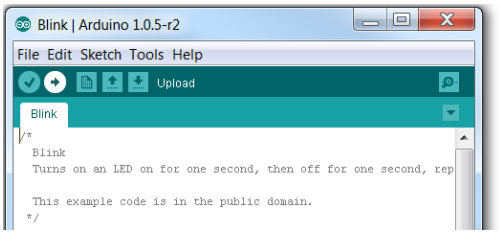


Fig. 6: Enviar o programa para o robô

Clicando em "***File -> Examples -> BnrOneA->...***" encontras todos os programas de exemplo fornecidos pela botnroll.com especificamente para o Bot'n Roll ONE A.

Em "***File -> Examples -> BnrOneA -> Basic ->...***" estão os programas básicos que têm como finalidade testar todo o *hardware* do robô. Deverás estudar e compreender bem estes pequenos programas!

Em "***File -> Examples -> BnrOneA -> Advanced -> ...***" estão programas mais avançados que só deverás estudar quando perceberes os mais simples.

Em "***File -> Examples -> BnrOneA -> Extra -> ...***" são os programas relacionados com os componentes extra que expandem o teu Bot'n Roll ONE A.

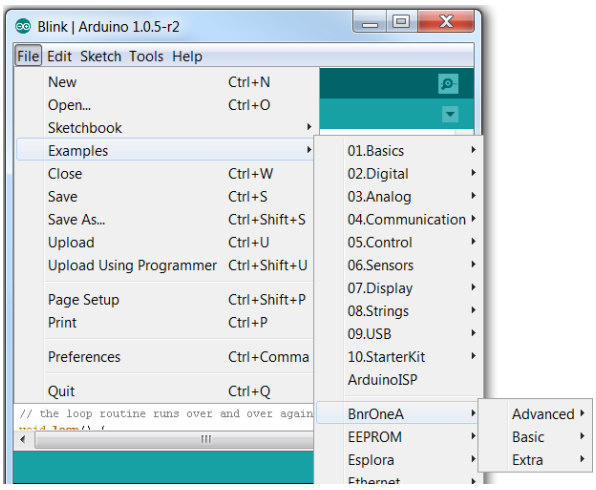


Fig. 7: Programas da biblioteca BnrOneA

