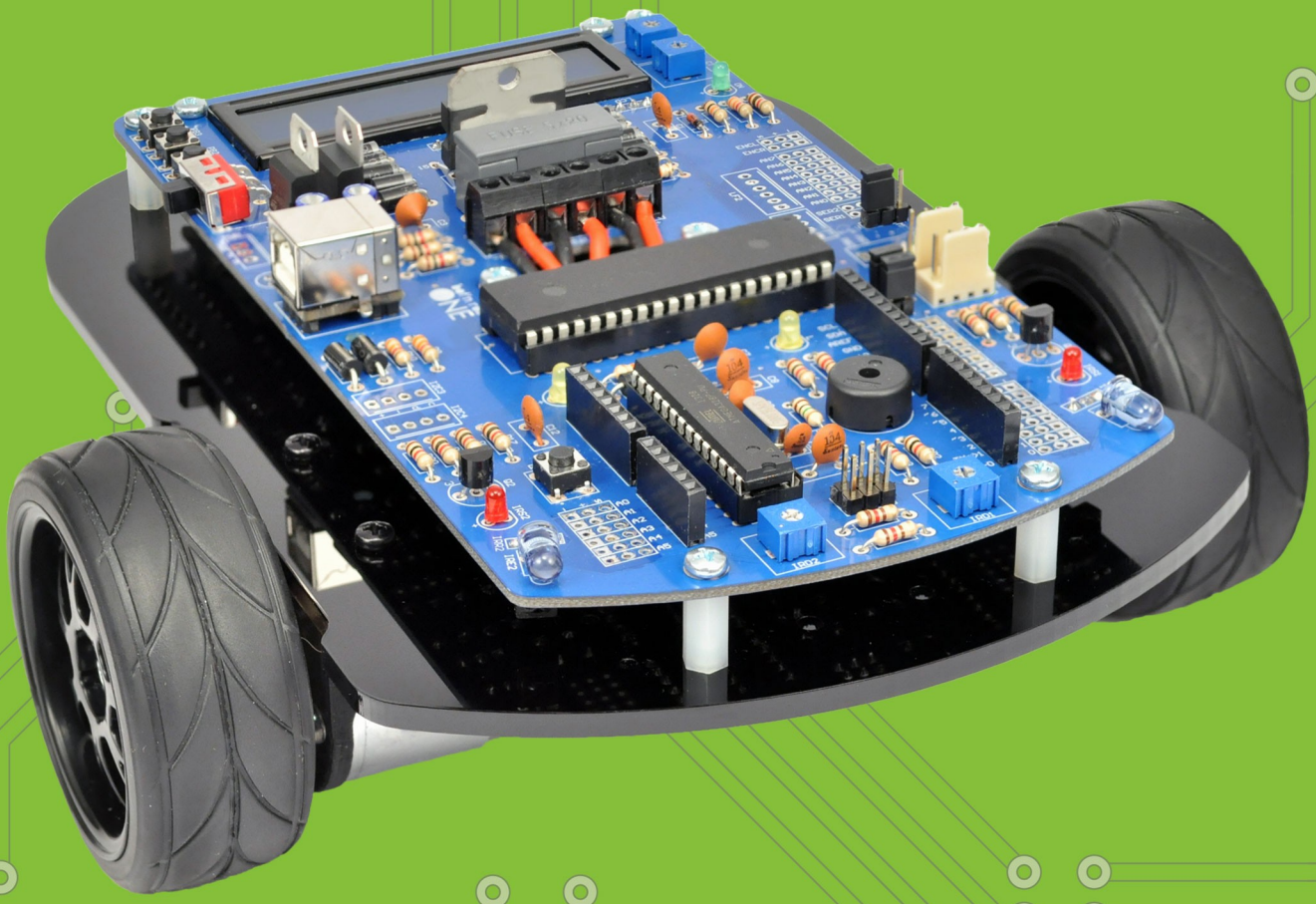
bot'n roll ONE

build your own robot



software user manual

www.botnroll.com

CONTENTS

Document Revision: May 29th, 2023

1. INTRODUCTION

The Bot'n Roll ONE A is programmed using C language on the Arduino IDE environment. The ATmega328 microcontroller on this robot has the Arduino Uno bootloader, therefore the robot is programmed as if it is an Arduino Uno.

The robot has a second micro-controller, a pre-programmed PIC18F45K22 with software developed by botnroll.com. On the Bot'n Roll ONE A it operates as a **slave** that executes the commands sent by the **master** ATmega328.

The two micro-controllers on the Bot'n Roll ONE A communicate with each other through the "**Serial Peripheral Interface**" SPI bus. The micro-controllers exchange information in a coordinated and well-defined way. A data transfer protocol was developed to allow communication between the master and the salve. The master uses a list of commands that corresponds to control instructions and each command generates a response from the slave. The list of commands and how data is transmitted between master and slave are defined in the BnrOneA library.

The BnrOneA library for Arduino makes it possible for the user to control the robot in a simple way, being only necessary to correctly use the library commands in the Arduino IDE. These commands are listed and explained in this manual.

Although the two micro-controllers can be programmed in C language, only the ATmega328 with Arduino bootloader is supposed to be programmed in a daily basis using the BnrOneA library.

The PIC18F45K22 can be programmed in C language using the Microchip's MPLABX IDE programming environment and XC8 compiler or other compatible software. However, this should only be done by advanced users, since programming the PIC18F45K22 to include a new feature, requires that you also update the BnrOneA library for the Arduino to be able to use the new functionality. Please contact botnroll.com if you like a new feature to be implemented on your Bot'n Roll ONE A!

1.1 PROGRAMMING THE BOT'N ROLL ONE A

To program the Bot'n Roll ONE A it is necessary that you have your computer prepared with all required tools, i.e.:

- VCP driver installed, the Bot'n Roll ONE A USB port driver (see ANNEX A);
- Arduino IDE installed (see ANNEX B);
- BnrOneA library installed on Arduino IDE (see ANNEX B).

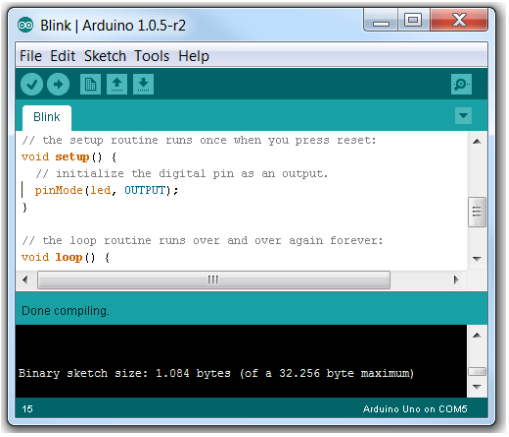
For detailed information about these tools installation, consult the annexes A and B on the final of this manual.

The C language is also one of the required tools for programming the Bot'n Roll ONE A. If you are still not very comfortable with C language, you can find the examples of the library to be a good guide for you to initiate on programming. The RoboParty presentations about C programming are a good guide also, and of course, there are thousands of internet websites that explain the C language!

1.1.1 Arduino IDE

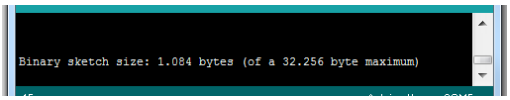
O Arduino programming environment contains a text editor to write code, an area for messages, a text console, a toolbar with the most important functions and a number of menus. It also allows to link to the Bot'n Roll ONE A Arduino hardware to transfer the code and communicate with the robot.

A program for Arduino is a "***sketch"***, it is written on the text editor and saved with extension ".ino" on your computer.



The messages area presents information about saving and exporting programs and presents the errors.

The console presents text messages with detailed information about the errors and other type of information.



On the bottom right corner of the window the information about the board to be programmed and the serial port in use is presented.

Fig. 1: Arduino IDE Modules

The main icon buttons on the toolbar and its functions are:

Verify*:* Checks for errors on the code.

Upload: Compiles the code and sends it to Arduino.

New: Creates a new sketch.

Open: Opens a sketch saved on the computer.

Save: Saves the **sketch**.

Serial Monitor: Opens the serial port communication for monitoring.

The serial monitor allows you to view data sent from the Arduino to the computer, also allows sending data from the computer to the Arduino. It is very useful in programming because here you can print text and variables values and thus it works as a "**debugging**" tool for your program. When you open the serial monitor your Arduino program restarts.

1.1.2 BnrOneA Library for Arduino

A library is a "pre-written" set of routines that you can include on your code and use on your program. To use the **BnrOneA** library you just need to include it in your code:

#include <BnrOneA.h>

And create an instance for the class BnrOneA:

BnrOneA one;

From now on you have access to all library functions that are preceded by the instance that you defined i.e.: **one.***library_function()*;

A library is normally created to manipulate data or hardware and has always at least two files, but on the Arduino case, there is still a third additional file with extension "**.txt"**.

- One file with extension "**.h**" ("***header***") which contains the list of all available functions, commands and library definitions;
- One file with extension "**.cpp**" ("***c++ source***") with the actual code for the functions presented on the header file.
- One ***keywords.txt*** file that allows Arduino IDE to identify the library functions and presenting them with a different color from the rest of the code.

The BnrOneA library was created to manipulate the PIC18F45K22 associated hardware and allows Arduino to interact with it through the SPI communication bus. The Arduino has access to all hardware and functionalities defined on the library and on the PIC18F45K22 software. The BnrOneA library and the PIC18F45K22 software were created "one for the other" and any change on one of them requires adjusting the other one.

All BnrOneA library functions are specified and explained on section 2 of this manual.

1.1.3 C Programming Language

The C language was developed in 1972 by Dennis Ritchie at Bell Labs in New Jersey. It emerged with the idea of being a powerful and fast language to be used in the UNIX operating system. Over time, it has been improved and updated showing up to be very robust and reliable, and it started to be used by other operating systems such as Windows, MacOS and Linux. It is in constant evolution since it emerged as the first version known as "**K & R C**". In 1989 emerged the first specification as a standard by the American Institute of patterns the "**ANSI C**". In 1990 emerged the "ISO C" by the International Organization for Standardization. In 1999, the "**C99**" standard emerged and the latest revision date of December 2011 to ISO / **IEC 9899:2011** better known as "C11".

All these updates and revisions are aimed at the use of the C language to develop programs for personal computers where memory and processing resources are not a constraint. For use in microcontrollers it is used a "lighter" version of the C language because memory and processing resources are limited. Thus, to successfully program your Bot'n Roll ONE A you need to know a few basic rules of the language for Arduino and how some commands run.

All programs for Arduino have two routines, or functions, which are mandatory. The "**setup()**" configuration routine is executed only once at the start of your program. Here you must write all the code to initialize variables, set input and output pin, set SPI communication, Serial, I2C, in fact all the necessary configurations.

After configuration, your program enters the **loop()** routine and stays there indefinitely. The term **loop** means cycle and in this case, it is an infinite loop when the program reaches the end of the cycle back to the beginning and start all over again! This is where you write the program and offspring intelligence to your robot!

C Programming is not explained in this manual, please refer to the examples of BnrOneA library and Arduino in general. All the code is properly commented and you have to try and test for yourself to understand how it works. However, we leave you some tips we have learned over time in **botnroll.com**:

- Create new programs from the basic examples. Try to join 3 or 4 features of the robot in the same program from the basic examples!
  - A program rarely works the first time! Do not lose heart, analyze the problem and solve it!
- Insert code systematically and test it as you go along to check if everything runs as expected.
- To get a working program you will need more time to test you rather than to write it!
  - Use tools such as debug LED, the serial monitor or the LCD to print the value of variables and check if the program passes a certain point of the code.
- Programming is like practicing a new sport. At the beginning, it is painful because you do not have the necessary physical condition, do not know the rules and you take time to understand it. By training, you practice and improve in all aspects. With your work, you will end up being part of the main team!

The Bot'n Roll ONE A allows interaction with a very wide range of hardware. There are extras, commonly called "shields" for Arduino to do just about everything you imagine and these are compatible with Bot'n Roll ONE A! All shields have libraries to help you use and integrate, and your imagination is the limit!

2. BNRONEA ARDUINO LIBRARY FUNCTIONS

The BnrOneA library has functions or routines that allow the Arduino (ATmega328) access to all peripherals controlled by the PIC18F45K22. These routines are divide in three groups: configuration, reading and writing. Many of these functions use **parameters/arguments** to exchange information, i.e., sending and/or receiving variable values.

An argument is any expression inside the parentheses of a function e.g.:

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

The **argument** is **sspin** and is transferred as a **parameter** during the execution of the **spiConnect** function.

A function can also return a value as the result of its execution:

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

The battery reading function returns the value of the battery as a result of its execution. The battery value is stored in the variable **battery**.

Following one can read the list of all the functions of the BnrOneA library extracted from the **BnrOneA.h** file.spiConnect(sspin)

BnrOneA library functions list

Setup routines	**Reading routines**
void spiConnect(byte sspin);	byte obstacleSensors();
void minBat(float batmin);	byte readIRSensors();
void obstacleEmitters(boolean state);	byte readRangeL();
void saveCalibrate(float batmin,byte speedL,byte speedR);	byte readRangeR();
	int readAdc(byte);
	int readAdc0();
Command routines	int readAdc1();
void servo1(byte position);	int readAdc2();
void servo2(byte position);	int readAdc3();
void led(boolean state);	int readAdc4();
void move(intspeedL,intspeedR);	int readAdc5();
void moveCalibrate(int speedL,int speedR);	int readAdc6();
void move1m(byte motor, int speed);	int readAdc7();
void movePID(int speedL,int speedR);	Int readButton();
void stop();	float readBattery();
void stop1m(byte motor);	int readEncL();
void brake(byte torqueL,bytetorqueR);	int readEncR();
void brake1m(byte motor, byte torque);	int readEncLInc();
void brake1m(byte motor);	int readEncRInc();
void resetEncL();	void readFirmware(byte*,byte*,byte*);
void resetEncR();	
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 SETUP ROUTINES

2.1.1 spiConnect(sspin)

Description:

Initializes the SPI communication bus by setting SCK, MOSI and SS pins as outputs. Places the SCK and MOSI pins at low state (0V) and SS in high state (5V). In Bot'n Roll ONE A, the SS pin "Slave Select" for SPI communication between the ATmega328 and PIC18F45K22, corresponds by default to the **digital output 2** but can be changed by removing the jumper SSP and making the desired connection. The native ATmega328 SS pin can be used for SPI communication with any shield that uses this link.

Parameters:

sspin: the digital output to be used as "*Slave Select*" on the SPI communication between the ATmega328 and the PIC18F45K22. (*byte*)

Returns:

Nothing

Example:

```
#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)

Description:

Defines the minimum value of battery voltage at which the robot motion blocks, and writes a warning message on the second row of the LCD. This setting value is written to EEPROM on PIC18F45K22 and is not lost when you shut down Bot'n Roll ONE A. During the writing on the EEPROM, the PIC18F45K22 is unreachable, therefore no commands must be sent during the process that lasts approximately 5ms. Whenever the battery voltage lowers below the minimum set, the robot does not respond to movement commands and shows a low battery message on line 2 of the LCD. This feature helps preserve battery life and is very useful in case of using lithium batteries that are destroyed if the voltage drops to the minimum value of safety.

Parameters:

batmin: The minimum voltage value allowed for the battery (float)

Returns:

Nothing

Example:

```
#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 saveCalibrate(bat,powerL,powerR)

Description:

Saves on EEPROM memory the battery voltage and motor power values acquired during motors calibration. Motors calibration improve robot motion at very low speeds and the battery discharge to have minimal influence in robot speed.

Parameters:

bat: the battery voltage value acquired during motors calibration *(float)*

powerL: left motor power *(byte)*

powerR: right motor power *(byte)*

Returns:

Nothing

Example:

```
...
void loop()
{
    one.saveCalibrate(14.2 , 62 , 62); // save motors calibration data in EEPROM
...
}
```

2.1.4 obstacleEmitters(state)

Description:

Controls the state of the LED infrared emitters. The infrared emitters can be switched off if necessary and normal detection of obstacles is disabled. The value 0 turns the emitters off and the value 1 enables the emitters.

Parameters:

state: the state to place the LED's (*boolean*)

Returns:

Nothing

Example:

```
#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 READING ROUTINES

2.2.1 obstacleSensors()

Description:

Returns the result read from the obstacles and there are four possible situations:

- **0**: there are no obstacles
- **1**: obstacle detected on the left sensor
- **2**: obstacle detected on the right sensor
- **3**: obstacles detected on both left and right sensors

The obstacles reading is performed every 20ms by the PIC18F45K22 and this function returns the result of the last reading. To make the reading of obstacles it is necessary that the infrared emitters be activated!

Parameters:

None

Returns:

The value read by the obstacles (*byte*)

Example:

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

2.2.2 readIRSensors()

Description:

Returns the actual state of the VISHAY TSSP4056 infrared sensors and there are four possible situations:

- **0**: both sensors on "*High*"
- **1**: left sensor "*High*" and right sensor "*Low*"
- **2**: left sensor "*Low*" and right sensor "*High*"
- **3**: both sensors "*Low*"

This function forces the PIC18F45K22 to make an instant reading to the state of the infrared sensors. Note that the VISHAY TSSP4056 sensors have the inverted state of the infrared transmitter signal.

Parameters:

None

Returns:

The actual state of the infrared sensors (*byte)*

Example:

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

2.2.3 readRangeL()

Description:

Returns the proximity value of an obstacle to the left infra-red sensor and varies from 0 to 25:

- **0**: no obstacle is detected.
- **25**: obstacle is very close to the sensor.

The returned value increases with the proximity distance to the obstacle.

Parameters:

None

Returns:

The proximity value of an obstacle to the left infra-red sensor (*byte)*

Example:

```
…
void loop()
{
    byte rangeL=one.readRangeL(); // read left obstacle sensor range
…
}
```

2.2.4 readRangeR()

Description:

Returns the proximity value of an obstacle to the right infra-red sensor and varies from 0 to 25:

- **0**: no obstacle is detected.
- **25**: obstacle is very close to the sensor.

The returned value increases with the proximity distance to the obstacle.

Parameters:

None

Returns:

The proximity value of an obstacle to the right infra-red sensor (*byte)*

Example:

```
…
void loop()
{
    byte rangeR=one.readRangeR(); // read right obstacle sensor range
…
}
```

2.2.5 readAdc(byte)

Description:

Returns the value of ADC conversion "Analog to Digital Conversion" associated with the PIC18F45K22 for the function specified in the parameter channel. The parameter defines which of the analog channels AN0 to AN7 the Bot'n Roll ONE A you want the conversion and accepts the following values:

- **0** (corresponds to channel AN0)
- **1** (corresponds to channel AN1)
- **2** (corresponds to channel AN2)
- **3** (corresponds to channel AN3)
- **4** (corresponds to channel AN4)
- **5** (corresponds to channel AN5)
- **6** (corresponds to channel AN6)
- **7** (corresponds to channel AN7)

The value obtained in the conversion varies between 0 and 1023 for the PIC18F45K22 since it has a 10bits ADC converter.

Parameters:

The ADC channel (*byte*)

Returns:

The value of the ADC conversion (int)

Example:

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

2.2.6 readAdcX()

Description:

Returns the value of ADC conversion "Analog to Digital Conversion" associated with the PIC18F45K22 from ANX channel of Bot'n Roll ONE A. The value obtained in the conversion varies between 0 and 1023 for the PIC18F45K22 since it has 10bits ADC converter.

The "X" letter corresponds to the ADC reading channel that is intended to be read and should be replaced by a number from 0 to 7.

Parameters:

None

Returns:

The value of the ADC conversion (int)

Example:

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

2.2.7 readButton()

Description:

Indicates which of the pushbuttons PB1, PB2 or PB3 is being pressed. The function returns one of the possible values, as a result:

- **0**: no button is pressed
- **1**: PB1 pressed
- **2**: PB2 pressed
- **3**: PB3 pressed

If more than one pushbutton are pressed at the same time, the lowest value is returned.

Parameters:

None

Returns:

Number of the Button pressed (*int*)

Example:

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

2.2.8 readBattery()

Description:

Reads the value of the battery voltage in Volts.

Parameters:

None

Returns:

The voltage of the battery (float)

Example:

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

```
}
```

2.2.9 readEncL()

Description:

Returns the read count of the left encoder and the encoder performs a reset, clearing the count value. This function is useful for measuring and controlling the speed of a wheel. If the speed of a wheel is constant, in well-defined time intervals we should obtain the same count values.

The encoder corresponds to a 16bits counter and the count varies between -32768 and 32767.

Parameters:

None

Returns:

The count of the left encoder (*int*)

Example:

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

2.2.10 readEncR()

Description:

Returns the read count of the right encoder and the encoder performs a reset, clearing the count value. This function is useful for measuring and controlling the speed of a wheel. If the speed of a wheel is constant, in well-defined time intervals we should obtain the same count values.

The encoder corresponds to a 16-bit counter and the count varies between -32768 and 32767.

Parameters:

None

Returns:

The count of the right encoder (*int*)

Example:

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

2.2.11 readEncLInc()

Description:

Returns the reading of the incremental counting for the left encoder. This function is useful for controlling the distance traveled by a wheel. Knowing that the increase of one unit of the value of the encoder corresponds to a certain distance, one can control the distance traveled by a wheel. No reset is carried out to the encoder value. The encoder corresponds to a 16-bit counter and the count is between -32768 and 32767. The user should monitor the encoder overflow.

Parameters:

None

Returns:

Incremental count of the left encoder (*int*)

Example:

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

2.2.12 readEncRInc()

Description:

Returns the reading of the incremental counting for the right encoder. This function is useful for controlling the distance traveled by a wheel. Knowing that the increase of one unit of the value of the encoder corresponds to a certain distance, one can control the distance traveled by a wheel. Not reset is carried out to the encoder value. The encoder corresponds to a 16-bit counter and the count is between -32768 and 32767. The user should monitor the situation of encoder overflow.

Parameters:

None

Returns:

Incremental count of the right encoder (*int*)

Example:

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

2.3 COMMAND ROUTINES

2.3.1 servo1(position)

Description:

Moves the servo connected to SER1. The function parameter defines the angular position, which varies from 0 to 180.

A standard servo position itself on a defined angle send through a function parameter, and the angle varies between 0º and 180º.

A continuous rotation servo varies its rotation speed according to the value sent through the function parameter. '0' corresponds to the maximum rotation in a certain direction, 180 corresponds to the maximum in the opposite direction and 90 corresponds to the servo being stopped.

Parameters:

position: The servo angular position (*byte*)

Returns:

Nothing

Example:

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

2.3.2 servo2(position)

Description:

Moves the servo connected to SER2. The function parameter defines the angular position, which varies from 0 to 180.

A standard servo position itself on a defined angle send through a function parameter, and the angle varies between 0º and 180º.

A continuous rotation servo varies its rotation speed according to the value sent through the function parameter. '0' corresponds to the maximum rotation in a certain direction, 180 corresponds to the maximum in the opposite direction and 90 corresponds to the servo being stopped.

Parameters:

position: The servo angular position (*byte*)

Returns:

Nothing

Example:

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

2.3.3 led(state)

Description:

Turns on or off the Bot'n Roll ONE A LED. The LED status is defined by the parameter sent to the function and has two possible states:

- **0**: LED off
- **1**: LED on

Parameters:

state: the LED state (*boolean*)

Returns:

Nothing

Example:

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

2.3.4 move(speedL,speedR)

Description:

Moves the Bot'n Roll ONE A motors. The parameters define the speed of each motor, left and right, which ranges from -100 to 100. The value -100 corresponds to the maximum speed in reverse, 100 corresponds to the maximum speed in the forward direction and 0 stops the motors.

Parameters:

speedL: left motor speed (*int*)

speedR: right motor speed (int)

Returns:

Nothing

Example:

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

2.3.5 moveCalibrate(powerL,powerR)

Description:

Power is applied to Bot'n Roll ONE A motors. The parameters define the amount of PWM power that is applied to each motor and values from to 0 to 100 are accepted. The value 100 corresponds to full power being applied to the motors (100% duty cycle) while 0 means no power is applied.

Parameters:

powerL: power for the left motor (*int*)

powerR: power for the right motor (*int*)

Returns:

Nothing

Example:

```
...
void loop()
{
    one.moveCalibrate (62,62);          // 62% PWM duty cycle power applied to both motors.
...
}
```

2.3.6 stop()

Description:

It stops Bot'n Roll ONE A both motors. It cuts the energy to the motors and they rotate freely until they stop, no breaking torque is applied.

Parameters:

None

Returns:

Nothing

Example:

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

2.3.7 brake(torqueL,torqueR)

Description:

Stop the Bot'n Roll ONE A motors applying braking torque. The braking power of each motor is defined by the function parameters and varies between 0 and 100. The value 0 corresponds to stop without braking torque. The value 100 corresponds to stop with maximum braking torque and the motor blocks instantly!

Parameters:

torqueL: left motor braking torque (*byte*)

torqueR: right motor braking torque (*byte*)

Returns:

Nothing

Example:

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

2.3.8 resetEncL()

Description:

Cleans the left encoder count forcing its value to become 0.

Parameters:

None

Returns:

Nothing

Example:

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

2.3.9 resetEncR()

Description:

Cleans the right encoder count forcing its value to become 0.

Parameters:

None

Returns:

Nothing

Example:

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

2.4 LCD WRITING ROUTINES

2.4.1 lcdX(string[])

Description:

Prints on the X line of the LCD an array of characters (string) or text in quotation marks ("text to send") sent as parameter. Parameters are accepted as the variable type (char) and (constchar). Maximum size in characters for a string or text in quotation marks is 17 that corresponds to 16 written characters on the LCD plus the terminating character '**\0**'.

The "X" letter represents the LCD line on which the user intends to write and must be 1 or 2.

Parameters:

string[]: array with the characters to be written on the LCD (*char*) or (*constchar*).

Returns:

Nothing

Example:

```
...
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)

Description:

Prints on the X line of the LCD a number or variable that is sent as a parameter. The number can be a variable type (int) (unsigned int) (long int) (double) or (float).

The "X" letter represents the LCD line on which the user intends to write and must be 1 or 2.

Parameters:

number: number or variable to print (*int*), (*unsigned int*), (*long int*), (*double*) or (*float*).

Returns:

Nothing

Example:

```
...
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)

Description:

Prints on the X line of the LCD the text and number sent as parameters. As text is accepted variable type (constchar), or text between quotation marks. The total number of characters to be printed should be no greater than 16. The number can be a variable type (int) (unsigned int) (long int) (double) or (float).

The "X" letter represents the LCD line on which the user intends to write and must be 1 or 2.

Parameters:

string[]: characters to be written on the LCD (*constchar*).

number: number or variable to print (*int*), (*unsigned int*), (*long int*), (*double*) or (*float*).

Returns:

Nothing

Example:

```
...
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)

Description:

Prints on the X line of the LCD numbers or variables sent as parameters. Numbers can be variables of type (int) or (unsigned int).

The "X" letter represents the LCD line on which the user intends to write and must be 1 or 2.

Parameters:

num1: number or variable to print (*int*) or (*unsigned int*).

num2: number or variable to print (*int*) or (*unsigned int*).

Returns:

Nothing

Example:

```
...
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)

Description:

Prints on the X line of the LCD numbers or variables sent as parameters. Numbers can be variables of type (int) or (unsigned int).

The "X" letter represents the LCD line on which the user intends to write and must be 1 or 2.

Parameters:

num1: number or variable to print (*int*) or (*unsigned int*).

num2: number or variable to print (*int*) or (*unsigned int*).

num3: number or variable to print (*int*) or (*unsigned int*).

Returns:

Nothing

Example:

```
...
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)

Description:

Prints on the X line of the LCD numbers or variables sent as parameters. Numbers can be variables of type (int) or (unsigned int).

The "X" letter represents the LCD line on which the user intends to write and must be 1 or 2.

Parameters:

num1: number or variable to print (*int*) or (*unsigned int*).

num2: number or variable to print (*int*) or (*unsigned int*).

num3: number or variable to print (*int*) or (*unsigned int*).

num4: number or variable to print (*int*) or (*unsigned int*).

Returns:

Nothing

Example:

```
...
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
...
}
```

ANNEX A: INSTALLING THE USB-SERIAL (RS232) CONVERTER VCP DRIVER

The driver allows your computer operating system to communicate with the Bot'n Roll ONE A.

To install the driver visit the Bot'n Roll ONE A support web page http://botnroll.com/onea/ and download the "**VCP Driver - Windows**" or "**VCP Driver - Mac OS X**" by clicking on the correct choice according to your operating system. After download, decompress the "*.zip*" file and run the application.

Every time you connect your robot to your computer using the USB cable a virtual COM port (VCP) is created, through which the communication between your Bot'n Roll ONE A and your PC is carried out. The programming application environment uses the port to communicate with the Bot'n Roll ONE A and therefore transfer your programs to your robot.

The USB-Serial converter used by the Bot'n Roll ONE A is a **PoUSB12** product from *PoLabs* and uses the **CP2102** ***Bridge*** device from *Silicon Labs*.

ANNEX B: ARDUINO PROGRAMMING ENVIRONMENT

The *software* used to program the robot is Arduino IDE. This application is necessary to edit the programs in C language. It is also used to transfer your programs to the Bot'n Roll ONE A.

B.1 ARDUINO IDE INSTALLATION

To install the Arduino IDE, visit the Bot'n Roll ONE A support web page http://botnroll.com/onea/ and click on "**Arduino IDE "** in the ***"Software | Drivers"*** section to download it according to your operating system.

After the download, decompress the .zip file and copy it to a folder of your choice on your computer.

This folder contains several sub-folders and files, as well as the "**arduino.exe"** application which is the executable that starts the Arduino IDE.

B.2 INSTALLING THE **BNRONEA** LIBRARY ON ARDUINO

Libraries are your programming working tools. The **BnrOneA** library developed by **botnroll.com** for the Arduino IDE contains all the required commands to control the robot. This library has to be installed on the Arduino IDE.

On the Bot'n Roll ONE A support web page http://botnroll.com/onea/, download the **BnrOneA.zip** file by clicking on "**Arduino Library**".

In **Arduino IDE** click the tab **"Sketch" --> "Include Library" --> "Add .ZIP Library…."**, select the **BnrOneA.zip** file and the library is automatically installed. Restart the **Arduino IDE** so the library becomes fully functional. Now you have all the required tools to program your **Bot'n Roll ONE A** with success!

B.3 CONFIGURING COMMUNICATION WITH THE ROBOT

Before proceeding with this step, make sure you have installed the VCP driver correctly. Connect the Bot'n Roll ONE A to your computer using the supplied USB cable. At this time, a COM port to communicate with the robot is automatically assigned.

Open the Arduino IDE, press the "***Tools -> Board***" option, and select the "**Arduino Uno**" board. The **Bot´n Roll ONE A** will be programmed as if it was an Arduino Uno.

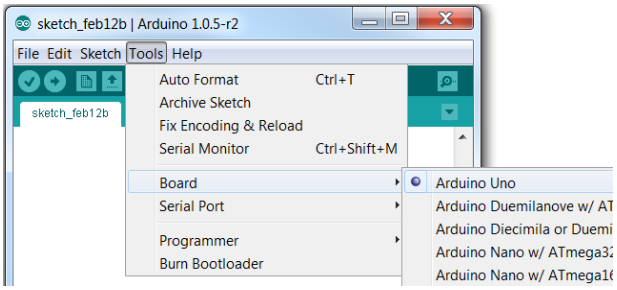


Fig. 2: Selecting the board to program

On the "***Tools -> Serial Port***" option, select the correct COM port given to the **Bot'n Roll ONE A**.

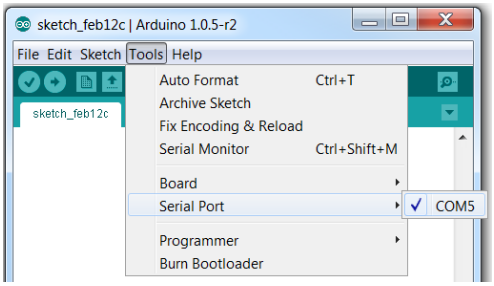


Fig. 3: Selecting the Serial port

Should no COM port be available, you probably did not install the USB-Serial converter VCP driver correctly.

Open the Windows device manager and search for the item "Ports (COM and LPT)". Expand the item, you will see all COM ports available.

"***Silicon Labs CP210x USB to UART Bridge***" is the designation which identifies the COM port that "talks" to Bot'n Roll ONE A (on this figure example, **COM5** port has been assigned).

Should the item "***Silicon Labs CP210x USB to UART Bridge***" not show up on the list, you have to install the VCP driver correctly.

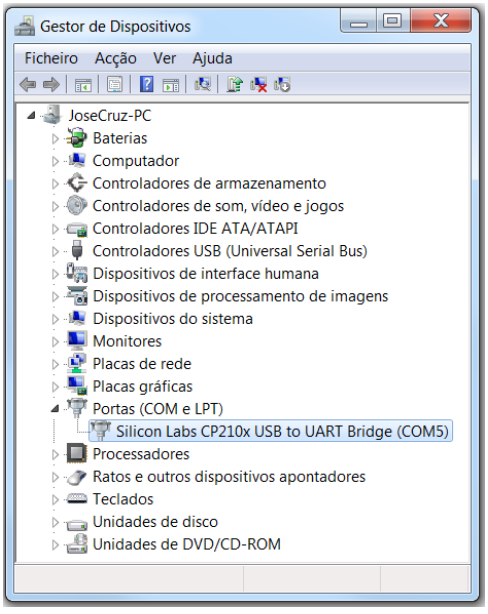


Fig. 4 COM ports on the device manager

B.4 LOADING A PROGRAM TO BOT'N ROLL ONE A

On the Arduino IDE application, you will find several example programs that you can load to your robot.

Click on "***File -> Examples -> 01.Basics -> Blink***" and a new window will show up with the code of this example.

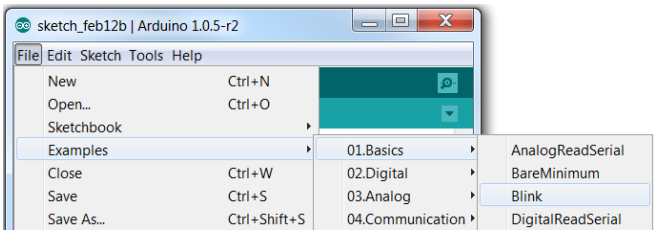


Fig. 5: Loading an example program

Click on "***File -> Upload***" or press the "arrow pointing right" icon to send the program to the robot. As soon as the upload finishes, you must see the yellow L LED flashing each second!

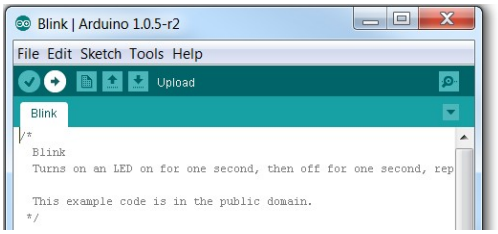


Fig. 6: Uploading a program to the robot

Clicking on "***File -> Examples -> BnrOneA -> ...***" you will find all the programs developed by botnroll.com specifically for Bot'n Roll ONE A.

On "***File -> Examples -> BnrOneA -> Basic -> ...***" you can find the simplest programs whose main purpose is to test all your robot hardware. You must study and understand well all these small programs!

On "***File -> Examples -> BnrOneA -> Advanced -> ...***" are available some more advanced programs which you might understand only after the simple ones.

On "***File -> Examples -> BnrOneA -> Extra -> ...***" you can find the programs related with the Bot'n Roll ONE A extras, which will make **Bot´n Roll ONE A** a more powerful robot.

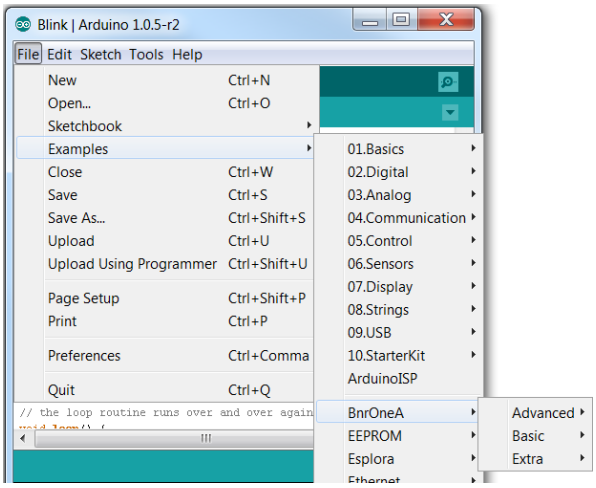


Fig. 7: Examples from the BnrOneA library