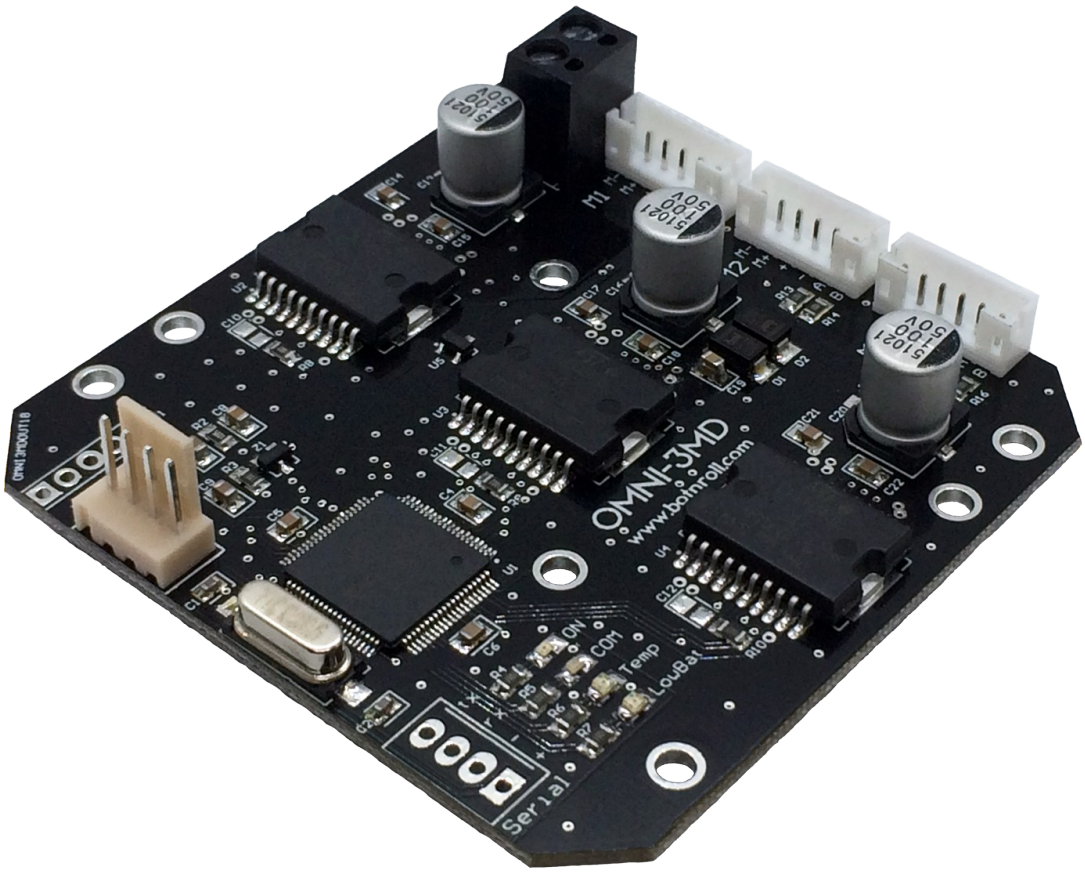
# OMNI-3MD

Three Motors Controller Board

*Arduino Library Available*




# Software User Manual

Firmware Version 1.90 – October 2019

# 1 Index

# 2 OMNI-3MD Overview

The OMNI-3MD control board is an I2C *SLAVE* device able to drive 3 DC motors from 8V to 50V and currents up to 5.6A RMS per motor. Using *encoders*, it can drive the motors in a PID control closed loop. A 16bits dsPIC processor running at 40MHz enables several different motor drives, namely:

- 3 motors omnidirectional drive (concentric, with the same distance to the centre and spaced 120º) with PID control.
- 2 motors differential drive with PID control using International System of Units (SI).
- Linear drive of 1, 2 or 3 motors with/without PID control.
- Positional drive of 1, 2 or 3 motors with PID control.

For more details related with the OMNI-3MD hardware functionalities check the hardware user manual available on http://botnroll.com/omni3md_en/

# 3 Communicating with OMNI-3MD

The communication with OMNI-3MD is carried out through an I2C bus. The control board uses by default the 7 bits I2C address 0x18 and receives commands sent from 100KHz to 400KHz. It also responds to the broadcast address 0x00.

The communication with the OMNI-3MD is carried out through streams sent to the I2C bus and can be write or read. A stream is made up by several bytes, which generally, follow the following format:

| ADDRESS | W/R | COMMAND | BYTE1 | BYTE2 | BYTE3 | BYTE4 | BYTE5 | … | BYTEn |
|---|---|---|---|---|---|---|---|---|---|
| 0x18 | W=0 | | **Data Bytes** | | | | | | |

**ADDRESS:** Is the I2C address field. This first byte contains the I2C address 7 bits and the eighth bit, the least significant, which defines if the stream is a writing or a reading one. If the address field least significant bit is 0 (zero), the stream is a writing one and the byte value is 0x30. If the least significant bit is 1 the stream is a reading one and the byte value is 0x31. In brief:

- 7 bits I2C Address: 0x18
- 8 bits I2C Address: 0x30 – stream for writing
- 8 bits I2C Address: 0x31 – stream for reading

**COMMAND:** It is the I2C command field. The command specifies the action the OMNI-3MD should perform.

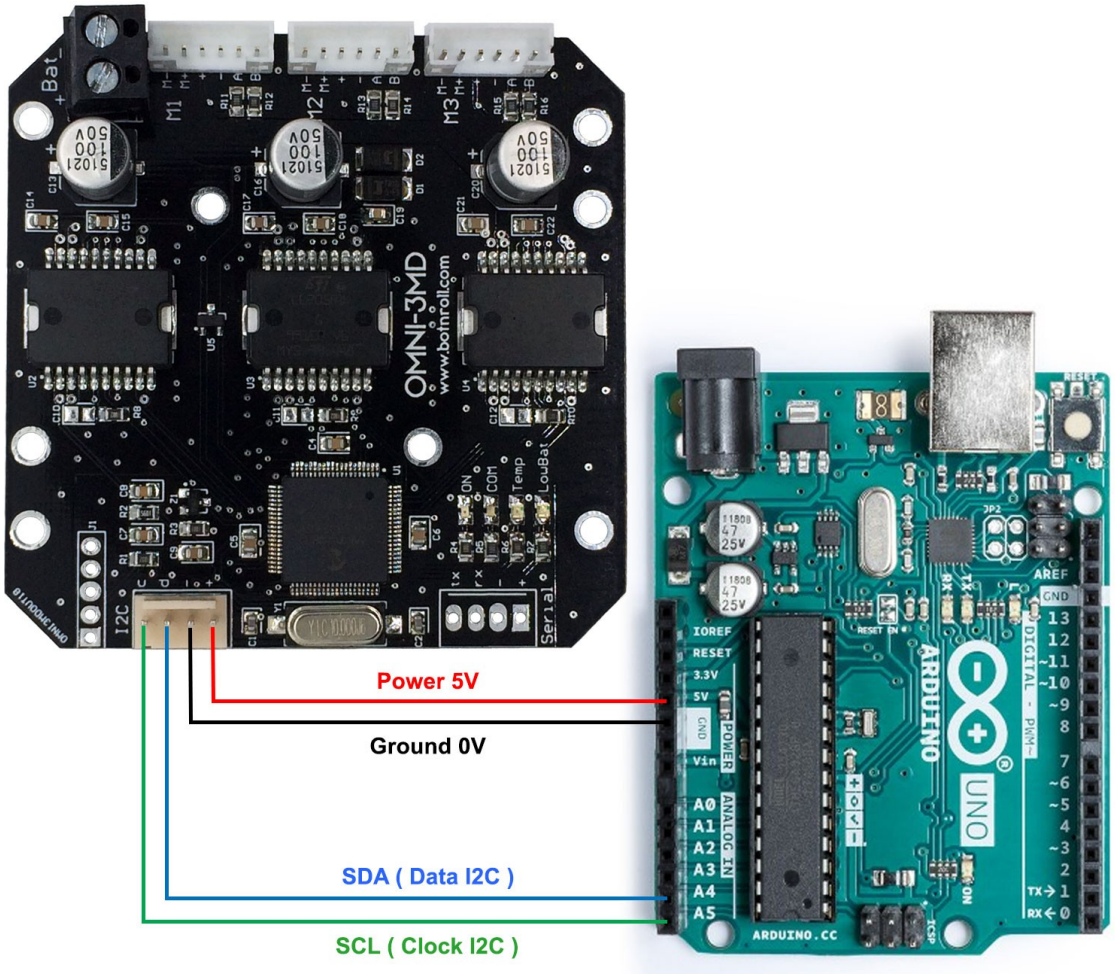
**Data Bytes:** These are the data bytes associated to each I2C command. Some commands are associated to two "**keys**", bytes 0x7A and 0x55. These keys are used to increase the communication reliability.

To easy the interaction with the OMNI-3MD a library was developed for C programming optimized for the Arduino IDE. The **BnrOmni library** is open-source and can be downloaded from the Omni-3MD control board support webpage:

*Download the BnrOmni Library:* http://www.botnroll.com/omni3md_en/

This library allows a higher-level usage of the Omni-3MD specially for the less experienced programmers to learn quickly and easily how to interact with the OMNI-3MD. There are examples on the library for all existing commands. The Arduino board and the Omni-3MD control board must be connected via I2C.

## 3.1 Example of I2C link between OMNI-3MD and Arduino



# 4 The BrnOmni Library

All existing commands on the BnrOmni library are hereby described. To every function, it is associated one command and the respective parameters represented by the I2C data bytes previously specified. There are setup functions, reading functions and movement functions. All these functions and respective commands are described on the BnrOmni.h file and source coded on the BnrOmni.cpp file of the BnrOmni Library.

## 4.1 Setup Functions

### 4.1.1 void i2cConnect(byte omniAddress);

This function should be performed only once, on the Arduino setup() routine and before any other functions from the BnrOmni library: it configures the Arduino I2C bus to communicate with the OMNI-3MD using the wire.h library.

The omniAddress parameter is the 8 bits I2C address for writing, by default 0x30.

Example:

```
#include <Wire.h>          //required by BnrOmni.cpp
#include <BnrOmni.h>
#define OMNI3MD_ADDRESS 0x30  //default factory address
BnrOmni omni;              //declaration of object variable to control the Omni3MD

void setup()
{
    omni.i2cConnect(OMNI3MD_ADDRESS);  //set i2c connection
}
```

This is a void function and therefore no value is returned.

### 4.1.2 void setI2cAddress(byte newAddress);

This function allows to change the Omni-3MD I2C address when required.

The newAddress parameter defines the new Omni-3MD address. The I2C protocol As specifications define as valid the 7 bits addresses between 0x08 and 0x77, i.e., it should be sent to the Omni-3MD even addresses of 8 bits between 0x10 (16) and 0xEE (238).

Example:

```
#include <BnrOmni.h>
BnrOmni omni;              //declaration of object variable to control the Omni3MD

void setup()
{
        ...
        omni.setI2cAddress(0x2E);
        ...
}
```

The I2C address received is stored in the EEPROM memory and therefore it is not lost when the Omni-3MD board is turned off. Writing this parameter on the EEPROM can take up to 5ms and during this time the Omni-3MD board does not respond to commands. The 10 ms delay used on the example takes the EEPROM writing into account and avoid the Arduino to send commands during that period.

This is a void function and therefore no value is returned.

### 4.1.3 void setI2cTimeout(byte timeout);

The I2C timeout is a protection mechanism against communication failures and therefore an important security measure. If the Omni-3MD does not receive I2C data for a certain time (*timeout*), it stops the motors avoiding possible damage and accidents that might occur on the system where the Omni-3MD is used. The timeout parameter is internally multiplied by 10 milliseconds and defines the I2C *timeout* time. Should we require a 500ms timeout, for example, the value 50 should be sent to this function.

This security mechanism can be disabled by sending the value 0 as parameter.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;            //declaration of object variable to control the Omni3MD

void setup()
{
        ...
        omni.setI2cTimeout(50);   //500ms timeout
        ...
}
```

Since timeout is one byte, it is possible to send values between 0 and 255. The maximum timeout is 2550ms, i.e. about two and a half seconds. The timeout value received is stored on the EEPROM memory and is not lost when the Omni-3MD is turned off. The writing of this parameter on the EEPROM takes about 5ms and during that time the Omni-3MD board does not respond to commands.

This is a void function and therefore no value is returned.

### 4.1.4 void calibrate(boolean way1, boolean way2, boolean way3);

Calibration is required when using motors with encoders. This routine moves the motors and analyses the encoders counting. During calibration, the maximum encoders counting is recorded and the PID control rate is automatically defined, which depending on the encoders could be 10, 20 or 40 times per second.
Quadrature encoders supply information about the direction of rotation. The Omni-3MD records the rotation direction during calibration as the reference for positive movement. When assembling your system, the motors rotation direction depends on the gearbox used and the electric wiring of the motor. The calibrate routine allows to define the positive direction in software, for any of the motors avoiding hardware changes, namely, changing the motors power supply wiring.

The parameters way1, way2 and way3 are used to define the direction of rotation of the calibration for each motor and these values can be either 0 or 1.
This is a void function and therefore no value is returned.
Example:

```
#include <BnrOmni.h>
BnrOmni omni;            //declaration of object variable to control the Omni3MD

void setup()
{
        ...
        omni.calibrate(1,0,1);
        delay(20000);          // wait 20s for calibration to end
        ...
}
```

The calibration must be performed on the initial system build up or every time there is a major change of the hardware, namely motors or batteries, should their Voltages be different. The calibration can be performed with the motors loaded or not loaded (ex.: wheels raised or standing on the floor), being the decision taken by the user according to the system. If the robot had been previously calibrated on the floor and if a major change on its weight occurred, it is recommends a new calibration. The values obtained on the calibration are stored on the EEPROM memory and are not lost when the Omni-3MD is turned off. The calibration process takes approximately 20 seconds, on which the motors must rotate without external interference. During calibration the yellow LED "I2C" remains on and the Omni-3MD does not respond to any I2C command.

**Note:** Never turn off or interfere with the system during calibration!

**Calibration and Omnidirectional Movement**

When using the Omni-3MD for Omnidirectional movement, the calibration routine parameters way1, way2 and way3 must be properly configured so that all motors rotate on the same direction and making the robot rotate on an anti-clockwise direction (CCW-*Counter clockwise*).

**Calibration and Differential Movement on the International System of Units (SI)**

When using the Omni-3MD for differential movement having the International System of Units (SI), use the outputs of motors 1 and 3. Configure the calibration routine parameters way1 and way3 in such way the robot moves forward.

### 4.1.5 void setPid(int Kp, int Ki, int Kd);

This routine allows to setup the Omni-3MD PID control parameters through the Kp, Ki and Kd variables which adjust on the Omni-3MD board the proportional, integral and differential gains, respectively.
The PID control gains must be adjusted in such way to obtain the required movement and small variations of gains can make a large difference on the movement. The PID gains take normally values between 0 and 1, but depending on the system, they can be over 1. The user must define the PID gains which best serve his system.
The Kp, Ki and Kd values must be positive and are internally divided by 1000 by the Omni-3MD board. To define a proportional gain of 0.65 on the Omni-3MD, for example, it should be sent as Kp parameter the value 650.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;           //declaration of object variable to control the Omni3MD

void setup()
{
        ...
        omni.setPid(650,450,250);
        ...
}
```

The values of Kp, Ki and Kd received are stored on EEPROM memory and are not lost when the Omni-3MD board is turned off. The writing of these 3 parameters on the EEPROM takes approximately 15ms and during that time the Omni-3MD does not respond to commands.
This is a void function and therefore no value is returned.

**Note:** Check the influence of the Kl parameter on the void set_ramp(int slope, int Kl) routine, in order to minimize overshoots in the special case of starting with initial velocity 0.

### 4.1.6 void setRamp(int slope, int Kl);

This function configures the acceleration ramp using the slope parameter. A perfectly adjusted acceleration ramp allows a smooth and efficient start with minimum slippage. The acceleration ramp implemented on the Omni-3MD board is a linear function with inclination adjusted by the slope parameter.

The slope parameter define the inclination of the acceleration ramp. Values accepted are from 0 to 100. Lower values provide smoother starts and higher values cause a more aggressive reaction.

The Kl parameter is the "Non Linearity" threshold, a mechanism intended to fight the non linearity of the DC motors. During calibration it is detected how much power is required to make the motors move. This power value is then multiplied by the threshold gain (Kl/1000) and used on the integral error of the PID uniquely on the first iteration of the control and when the system starts from the initial velocity of 0. This way the motors starting time is reduced and the performance of the PID control improves substantially, reducing overshoots. When adjusting the PID control it should be taken into account this parameter to optimize the starting with initial velocity of 0.
The Kl values should be positive and between 0 to 1000.
A Kl value of 0 disables this mechanism.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;            //declaration of object variable to control the Omni3MD

void setup()
{
        ...
        omni.setRamp(35,950);
        ...
}
```

The slope and Kl values received are stored in EEPROM memory and the information is not lost when the Omni-3MD board is powered off. Writing these 2 parameters on the EEPROM takes about 10ms and during this time the Omni-3MD board does not respond to commands.
This is a void function and therefore no value is returned.

### 4.1.7 void setEncPrescaler(byte encoder, byte value);

With this function the user defines the prescaler associated to each encoder. The prescaler is a factor that defines how the count of the pulses on the encoder transforms on the incremental count. On each cycle of the PID control the real count of the encoders is transformed on the incremental count having in mind the prescaler selected and the direction of movement.

- If a motor moves in the positive direction, the real value of the encoder will be added to the incremental count having in mind the defined prescaler.
- If a motor moves in the positive direction, the real value of the encoder will be subtracted to the incremental count having in mind the defined prescaler.

When a particular motor moves in the positive direction:
- A prescaler of 1 sets the actual value encoder is added directly to the incremental count every PID control loop.
- A prescaler of 10 defines that for each 10 pulses of the encoder actual count, one unit is added to the incremental count, every PID control loop.
- A prescaler of 1000 defines that for each 1000 pulses of the encoder actual count, one unit is added to the incremental count, every PID control loop.

The encoder parameter defines the encoder to which will change the prescaler. The valid values for this parameter are 1, 2 and 3 which correspond to motors 1, 2 and 3 respectively.

The value parameter defines the prescaler to be associated to the encoder. Valid values for this parameter:

- 0: defines a prescaler of 1
- 1: defines a prescaler of 10
- 2: defines a prescaler of 100
- 3: defines a prescaler of 1000
- 4: defines a prescaler of 10000

Example:

```
#include <BnrOmni.h>
BnrOmni omni;              //declaration of object variable to control the Omni3MD
#define M1  1          //Motor1
#define M2  2          //Motor2
#define M3  3          //Motor3

void setup()
{
   omni.setEncPrescaler(M1, 0);     //sets Motor1 prescaler to 1
   omni.setEncPrescaler(M2, 2);     //sets Motor2 prescaler to 100
   omni.setEncPrescaler(M3, 3);     //sets Motor3 prescaler to 1000;
}
```

The value value received is stored in EEPROM memory and is not lost when the Omni-3MD board is turned off. The writing on this parameter on the EEPROM takes about 5ms and during this time the Omni-3MD does not respond to commands.
This is a void function and therefore no value is returned.

### 4.1.8 void setDifferential(double axis_radius, double whell_radius, double gearbox_factor, double encoder_cpr);

This configuration is necessary when differential movement is required using International System of Units (SI). On this type of movement the units used are linear velocity in metres per second (m/s) and angular velocity (rotation) in radians per second (rad/s).
For the movement to be performed correctly, it is necessary to configure some parameters which correspond to physical characteristics of the robot:

axis_radius : Radius from the axis of rotation of the robot, i.e., half the distance between the wheels. The value must be entered in millimetres.

whell_radius : Radius of the drive wheels in millimetres.

gearbox_factor : Factor reduction gearbox DC motor. If the gearbox has a reduction ratio of 50:1, i.e., every 50 revolutions of the engine corresponds to one turn of the wheel, the value 50 must be entered.

encoder_cpr : The number of pulses generated by the encoder quadrature for a DC motor revolution (not the wheel).
Example:

```
#include <BnrOmni.h>
BnrOmni omni;            //declaration of object variable to control the Omni3MD

void setup()
{
  ...
    omni.setDifferential(97.5,37.5,29,60);
  ...
}
```

The axis_radius, whell_radius, gearbox_factor and encoder_cpr values are stored in EEPROM memory and the information is not lost when the Omni-3MD board is turned off. The writing of these 4 parameters on the EEPROM takes about 20ms and during this time the Omni-3MD board does not respond to commands.
This is a void function and therefore no value is returned.

## 4.2 Reading Routines

### 4.2.1 float readTemperature();

Reads the temperature of the Omni-3MD board. It is returned the temperature in Celsius degrees which should be stored in a float type variable.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;              //declaration of object variable to control the Omni3MD
float temperature=0.0;  // temperature reading

void loop()
{
    temperature=omni.readTemperature();  // read temperature value
}
```

### 4.2.2 float readBattery();

Reads the supply voltage of the motors on the Omni-3MD, which is usually provided by a battery. It returns the voltage in volts (V) to be stored in a variable of type **float**.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
float battery=0.0;      // battery reading

void loop()
{
    battery=omni.readBattery();        // read battery value
}
```

### 4.2.3 float readFirmware();

Reads the software version of the Omni-3MD. The software version has 3 fields and at the date of writing this manual, the 1.90 was the latest version. The parameters of this function receive the values through three byte pointers.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
float firmware =0.0;    // the firmware version

void loop()
{
    firmware=omni.readFirmware(); // read firmware version value
}
```

### 4.2.4 byte readControlRate();

Reads the rate control defined by the calibration routine Omni-3MD. This routine returns one byte.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
byte ctrl_rate=0;       // the control rate for your motors defined at calibration

void loop()
{
    ctrl_rate=omni.readControlRate();  // read the control rate value
}
```

### 4.2.5 int readEnc1Max();

Reads the maximum pulse values of encoder1 detected during calibration, for the control rate defined. This routine returns a variable of type 16bits integer.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc1_max;       // maximum count for encoder1 at calibration, for the defined control rate

void loop()
{
    enc1_max=omni.readEnc1Max();      // read encoder1 maximum value at calibration
}
```

### 4.2.6 int readEnc2Max();

Reads the maximum pulse values of encoder2 detected during calibration, for the control rate defined. This routine returns a variable of type 16bits integer.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc2_max;       // maximum count for encoder2 at calibration, for the defined control rate

void loop()
{
    enc2_max=omni.readEnc2Max();      // read encoder2 maximum value at calibration
}
```

### 4.2.7 int readEnc3Max();

Reads the maximum pulse values of encoder3 detected during calibration, for the control rate defined. This routine returns a variable of type 16bits integer.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc3_max;       // maximum count for encoder3 at calibration, for the defined control rate

void loop()
{
    enc3_max=omni.readEnc3Max();      // read encoder3 maximum value at calibration
}
```

### 4.2.8 int readEnc1();

Reads the value of the incremental count from encoder1. Each PID control loop, the incremental count is updated taking into account the actual encoder count, the prescaler and the direction of movement. This count may, for example, represent an offset in space and the user can use this information to control the movement of his robot / system.

This routine returns an 16bit integer ranging between -32768 and 32767.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc1=0;         // encoder1 reading, this is the encoder incremental count for the defined prescaler

void loop()
{
    enc1=omni.readEnc1();            // read encoder1 incremental value
}
```

### 4.2.9 int readEnc2();

Reads the value of the incremental count from encoder2. Each PID control loop, the incremental count is updated taking into account the actual encoder count, the prescaler and the direction of movement. This count may, for example, represent an offset in space and the user can use this information to control the movement of his robot / system.

This routine returns an 16bit integer ranging between -32768 and 32767.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc2=0;        // encoder2 reading, this is the encoder incremental count for the defined prescaler

void loop()
{
    enc2=omni.readEnc2();          // read encoder2 incremental value
}
```

### 4.2.10 int readEnc3();

Reads the value of the incremental count from encoder3. Each PID control loop, the incremental count is updated taking into account the actual encoder count, the prescaler and the direction of movement. This count may, for example, represent an offset in space and the user can use this information to control the movement of his robot / system.

This routine returns an 16bit integer ranging between -32768 and 32767.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc3=0;        // encoder3 reading, this is the encoder incremental count for the defined prescaler

void loop()
{
    enc3=omni.readEnc3();          // read encoder3 incremental value
}
```

### 4.2.11 int readLim1();

Reads the threshold value for motor 1 acquired during calibration.

This routine returns an 16bit integer.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int limiar1;       // Necessary motor power to start movement. Acquired during calibration

void loop()
{
    enc3=omni.readLim1();          // read limiar value for motor 1
}
```

### 4.2.12 int readLim2();

Reads the threshold value for motor 2 acquired during calibration.

This routine returns an 16bit integer.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int limiar2;       // Necessary motor power to start movement. Acquired during calibration

void loop()
{
    enc3=omni.readLim2();          // read limiar value for motor 2
}
```

### 4.2.13 int readLim3();

Reads the threshold value for motor 3 acquired during calibration.

This routine returns an 16bit integer.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int limiar3;       // Necessary motor power to start movement. Acquired during calibration

void loop()
{
    enc3=omni.readLim3();          // read limiar value for motor 3
}
```

### 4.2.14 void readEncoders(int*,int*,int*);

Reads the incremental count value of the encoders of the 3 motors. This function optimizes the communication I2C by transmitting the three encoders values in one go. The parameters of this function receive the values through three pointers to 16bits integers.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc1=0;        // encoder1 reading, this is the encoder incremental count for the defined prescaler
int enc2=0;        // encoder2 reading, this is the encoder incremental count for the defined prescaler
int enc3=0;        // encoder3 reading, this is the encoder incremental count for the defined prescaler

void loop()
{
    omni.readEncoders(&enc1,&enc2,&enc3); // read incremental value for 3 encoders at once
}
```

This is a void function and therefore no value is returned.

### 4.2.15 void readMovData(int*,int*,int*,float*,float*);

Reads the parameters that are directly related to the movement, all in one go. In one I2C transmission reads the incremental value of the 3 encoders, the value of motor supply voltage and temperature of the Omni-3MD board. All these values are placed in the variables (parameters) by means of pointers. This routine greatly reduces the communication time compared to the individual reading of these same values, optimizing thus the use of the I2C bus.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc1=0;          // encoder1 reading, this is the encoder incremental count for the defined prescaler
int enc2=0;          // encoder2 reading, this is the encoder incremental count for the defined prescaler
int enc3=0;          // encoder3 reading, this is the encoder incremental count for the defined prescaler
float battery=0.0;      // battery reading
float temperature=0.0;   // temperature reading

void loop()
{
   omni.readMovData(&enc1,&enc2,&enc3,&battery,&temperature);
}
```

This is a void function and therefore no value is returned.

### 4.2.16 void readAllData (int*,int*,int*,float*,float*,byte*,byte*,byte*,byte*,int*,int*,int*);

Reads all possible parameters available from the Omni-3MD board. In a single I2C transmission it reads the incremental value of the three encoders, the values of all motors supply voltages, the temperature of the Omni-3MD board, the three fields of the firmware version, the control rate and the pulses maximum values of the encoder detected during calibration. This routine greatly reduces the communication time compared to the individual reading of these same values, optimizing thus the use of the I2C bus.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int enc1=0;          // encoder1 reading, this is the encoder incremental count for the defined prescaler
int enc2=0;          // encoder2 reading, this is the encoder incremental count for the defined prescaler
int enc3=0;          // encoder3 reading, this is the encoder incremental count for the defined prescaler
float battery=0.0;      // battery reading
float temperature=0.0;   // temperature reading
byte firm_rel=0;     // the firmware release field
byte firm_int=0;     // the firmware intermediary field
byte firm_dev=0;     // the firmware development field
byte ctrl_rate=0;       // the control rate for your motors defined at calibration (in times per second)
int enc1_max;        // maximum count for encoder 1 at calibration, for the defined control rate
int enc2_max;        // maximum count for encoder 2 at calibration, for the defined control rate
int enc3_max;        // maximum count for encoder 3 at calibration, for the defined control rate

void loop()
{


omni.readAllData(&enc1,&enc2,&enc3,&battery,&temperature,&firm_rel,&firm_int,&firm_dev,&ctrl_rate,&enc1_max,&enc2_max,&enc3
_max);

}
```

This is a void function and therefore no value is returned.

## 4.3 Movement routines

### 4.3.1 void movOmni(byte linear_speed, int rotational_speed, int direction);

This routine sends an order of Omnidirectional movement for the Omni-3MD board. This type of movement is only possible in a system with encoders and is subject to closed-loop control of PID.

The parameters sent to the Omni-3MD are:

linear_speed : the value of the movement linear velocity in percentage of maximum speed, which varies between 0 and 100;

rotational_speed : the value of the angular velocity as a percentage of the maximum velocity, which varies between -100 and 100. The sign indicates the direction of rotation and the value 0 means no rotation is applied to the movement;

direction : the value of motion direction in degrees between 0 and 360;

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
 int lin_speed=50;
 int rot_speed=0;
 int dir=180;

void loop()
{
   omni.movOmni(lin_speed,rot_speed,dir);   //move motors
   delay(100);                 // The time for the PID control rate
}
```

The delay immediately after the movement command is not necessary, however, the last command received by the OMNI-3MD will be performed on the next PID control loop. Depending on the control rate, the time to perform the movement command is 25ms, 50ms or 100ms.

This is a void function and therefore no value is returned.

### 4.3.2 void movDifSi(double linear_speed, double rotational_speed);

This routine sends a movement command to the Omni-3MD board, of type differential using International System of Units (SI). This type of movement is possible only in a system with encoders and is subject to closed-loop PID control.

The parameters sent to the Omni-3MD board are:
linear_speed : value of the linear velocity in meters per second (m/s);
rotational_speed : value of the angular speed in meters per second (rad/s);

For a certain motor, the 0 velocity value corresponds to stop the motor. Negative values make the motor to rotate on the negative direction (defined in the calibration) and positive values make the motor rotate in the positive direction.
Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
double lin_speed_si=0.5;   //Linear speed in m/s
double rot_speed_si=0.0;   //Rotational speed in rad/s
void loop()
{
   omni.movDifSi(lin_speed_si,rot_speed_si);   //Move forward
   delay(100);
}
```

The delay immediately after the movement command is not necessary, however, the last command received by the OMNI-3MD will be performed on the next PID control loop. Depending on the control rate, the time to perform the movement command is 25ms, 50ms or 100ms.
This is a void function and therefore no value is returned.

### 4.3.3 void mov3mPid(int speed1, int speed2, int speed3);

This routine sends a command to the Omni-3MD for independent motors drive. This type of movement is possible only in a system with encoders and is subject to closed-loop PID control.

The parameters sent to the Omni-3MD board are:

speed1 : velocity value for motor 1, as a percentage of the maximum velocity, which varies between -100 and 100;

speed2 : velocity value for motor 2, as a percentage of the maximum velocity, which varies between -100 and 100;

speed3 : velocity value for motor 3, as a percentage of the maximum velocity, which varies between -100 and 100;

For a certain motor, the 0 velocity value corresponds to stop the motor. Negative values make the motor to rotate on the negative direction (defined in the calibration) and the maximum velocity value is -100. Positive values make the motor rotate in the positive direction (defined in the calibration) and the maximum velocity value is 100.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int speed1=20;
int speed2=40;
int speed3=60;
void loop()
{
   omni.mov3mPid(speed1,speed2,speed3);   // move motors with PID control
   delay(100);
}
```

The delay immediately after the movement command is not necessary, however, the last command received by the OMNI-3MD will be performed on the next PID control loop. Depending on the control rate, the time to perform the movement command is 25ms, 50ms or 100ms.

This is a void function and therefore no value is returned.

### 4.3.4 void mov1mPid(byte motor, int speed);

This routine sends the Omni-3MD an order for an independent motor movement. This type of movement is only possible in a system with encoders and is subject to a closed-loop PID control.

The parameters for this routine are:
motor : The motor to move: values between 1 and 3;
speed : velocity value for the motor, in percentage of the maximum velocity, varies between -100 and 100;

For a certain motor, the 0 velocity value corresponds to stop the motor. Negative values make the motor to rotate on the negative direction (defined in the calibration) and the maximum velocity value is -100. Positive values make the motor rotate in the positive direction (defined in the calibration) and the maximum velocity value is 100.
Example:

```
#include <BnrOmni.h>
#define M2  2          //Motor2
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int speed=20;

void loop()
{
   omni.mov1mPid(M2,speed);   // move motor 2 at the desired speed
   delay(100);
}
```

The delay immediately after the movement command is not necessary, however, the last command received by the OMNI-3MD will be performed on the next PID control loop. Depending on the control rate, the time to perform the movement command is 25ms, 50ms or 100ms.
This is a void function and therefore no value is returned.

### 4.3.5 void mov3m(int speed1, int speed2, int speed3);

This routine sends the Omni-3MD an order for three independent drive motors. This type of movement is not subject to PID control in closed loop.

This routine parameters sent to the Omni-3MD are:

speed1 : velocity value for motor 1, as a percentage of the maximum velocity, which varies between -100 and 100;

speed2 : velocity value for motor 2, as a percentage of the maximum velocity, which varies between -100 and 100;

speed3 : velocity value for motor 3, as a percentage of the maximum velocity, which varies between -100 and 100;

For a certain motor, the 0 velocity value corresponds to stop the motor. Negative values make the motor to rotate on the negative direction (defined in the calibration) and the maximum velocity value is -100. Positive values make the motor rotate in the positive direction (defined in the calibration) and the maximum velocity value is 100.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int speed1=20;
int speed2=40;
int speed3=60;
void loop()
{
   omni.mov3m(speed1,speed2,speed3);   // move motors with no PID control
   delay(100);
}
```

The delay immediately after the movement command is not necessary, however, the last command received by the OMNI-3MD will be performed on the next PID control loop. Depending on the control rate, the time to perform the movement command is 25ms, 50ms or 100ms.

This is a void function and therefore no value is returned.

### 4.3.6 void mov1m(byte motor, int speed);

This routine sends the Omni-3MD an order for an independent movement of a motor. This type of motion is not subject to PID control in closed loop.

The parameters for this routine are:
motor : The motor to move: values between 1 and 3;
speed : velocity value for the motor, in percentage of the maximum velocity, varies between -100 and 100;

For a certain motor, the 0 velocity value corresponds to stop the motor. Negative values make the motor to rotate on the negative direction (defined in the calibration) and the maximum velocity value is -100. Positive values make the motor rotate in the positive direction (defined in the calibration) and the maximum velocity value is 100.

Example:

```
#include <BnrOmni.h>
#define M2  2          //Motor2
BnrOmni omni;      //declaration of object variable to control the Omni3MD
int speed=20;

void loop()
{
   omni.mov1m(M2,speed);   // move motor 2 at the desired speed
   delay(100);
}
```

The delay immediately after the movement command is not necessary, however, the last command received by the OMNI-3MD will be performed on the next PID control loop. Depending on the control rate, the time to perform the movement is 25ms, 50ms or 100ms.
This is a void function and therefore no value is returned.

### 4.3.7 void setEncValue(byte encoder, int encValue);

This routine allows the user to predefine a value for the incremental count of the encoders.

The parameter encoder defines which encoder will be subject to this change in the incremental count. The valid values for this parameter are 1, 2 and 3 which correspond to motors 1, 2 and 3 respectively.

The encValue parameter defines the value to pre define the incremental count of the encoder. Since this is a 16bits type variable, the values can vary between -32768 and 32767.

Example:

```
#include <BnrOmni.h>
#define M1  1          //Motor1
#define M2  2          //Motor2
#define M3  3          //Motor3
BnrOmni omni;              //declaration of object variable to control the Omni3MD
...
void loop()
{
   omni.setEncValue(M1,0);     // resets to zero the encoder value [byte encoder, word encValue]
   omni.setEncValue(M2,100);
   omni.setEncValue(M3,25000);
...
}
```

This is a void function and therefore no value is returned.

### 4.3.8 void movPosition(byte motor, int speed, unsigned int encPosition);

This routine sends the Omni-3MD an order to position a motor. Causes a particular motor to move from an initial position (encoder value) to a final position (encoder value) set by the user. This type of movement is only possible in a system with encoders and is subject to a closed-loop PID control.
The routines void **setEncValue**(byte encoder, int encValue) and void **setEncPrescaler**(byte encoder, byte value) are directly related with this type of movement so that its understanding is essential.

The motor parameter defines which motor is to be moved, being possible the values between 1 and 3.

The speed parameter defines the motor velocity, in percentage of the maximum velocity, and valid values are between -100 and 100.
For a certain motor, the 0 velocity value corresponds to stop the motor. Negative values make the motor to rotate on the negative direction (defined in the calibration) and the maximum velocity value is -100. Positive values make the motor rotate in the positive direction (defined in the calibration) and the maximum velocity value is 100.

The encPosition parameter defines the final position (incremental count) to reach. Valid values are between -32768 and 32767, since the incremental count is stored in a 16 bits int type variable.

Example:

```
#include <BnrOmni.h>
#define M1  1          //Motor1
BnrOmni omni;      //declaration of object variable to control the Omni3MD
//Variables for motion control
int speed1=90;
int preset1=100;
int pos1=3200;

void setup()
{
   ...
   omni.setEncPrescaler(M1, 1);     //sets the prescaler to 10
   ...
}

void loop()
{
   //Encoder preset
   omni.setEncValue(M1,preset1);     // resets to zero the encoder value [byte encoder, word encValue]

//Send movement instructions
   omni.movPosition(M1,speed1,pos1);  //
   while(enc1<pos1)
   {
    omni.readEncoders(&enc1,&enc2,&enc3); // read all encoders at once (in a single I2C request)
    Serial.print("PositionM1:");  Serial.println(enc1); //Print encoder position
    delay(100);
   }
}
```

The delay immediately after the movement command is not necessary, however, the last command received by the OMNI-3MD will be performed on the next PID control loop. Depending on the control rate, the time to perform the movement command is 25ms, 50ms or 100ms.

This is a void function and therefore no value is returned.

### 4.3.9 void savePosition();

This routines stores the incremental value of the encoders in EEPROM memory so that they are not lost when the OMNI-3MD is turned off. At the end of a positional drive it is possible to store the motors position by calling this function, should the system be turned off voluntarily or due to a power failure.

**Note:** This routine should not be used while motors move, because the EEPROM writing temporarily disables PID control and encoders count, compromising position.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;              //declaration of object variable to control the Omni3MD
...
void loop()
{
  omni.savePosition();  //save encoders positional data to EEPROM
}
```

This is a void function and therefore no value is returned.

### 4.3.10 void stop();

This routine sends the Omni-3MD an order to stop all motors. The motor stop without torque retention, i.e., rotating freely.

Example:

```
#include <BnrOmni.h>
BnrOmni omni;              //declaration of object variable to control the Omni3MD
...
void loop()
{
  omni.stop();  //stop all motors
}
```

This is a void function and therefore no value is returned.