Dynamic Motion

Dynamic Motion motor driver boards and programmable LED driver

Software manual 3.03






Subject to change without prior notice
Dynamic Motion SA
105, rue Fritz Courvoisier
2300 La Chaux-de-Fonds
Switzerland

welcome@dynamicmotion.ch
www.dynamicmotion.ch
Phone.: +41 32 968 64 50
Fax: +41 32 968 64 51

Table of content

Dynamic Motion SA
105, rue Fritz Courvoisier
2300 La Chaux-de-Fonds
Switzerland
welcome@dynamicmotion.ch
www.dynamicmotion.ch
Phone.: +41 32 968 64 50
Fax: +41 32 968 64 51

Part 1 introduction

Application

This manual is applicable to the following boards:

Boards with BASIC version 2.x

- Tinaxis Plus BL200
- Tinaxis Plus BL201
- Tinaxis Plus BL960
- Tinaxis Plus BL120
- Tinaxis DC200

Boards with BASIC version 1.x

- Tinaxis Plus BL60
- Tinaxis Plus BL57i150
- Tinaxis Plus BL57i650
- Tinaxis Plus BL86i650
- Tinaxis Plus STP60
- Tinaxis Plus STP400

LED controllers

- ANYLED470
- ANYLED200

Customer specific electronics

- Many models, when DM BASIC is present, this manual is applicable

Preface

This manual is the software description for the programmers of the Dynamic Motion products, based on DM-Basic and DM-Remote languages. It must be completed by the register description specific to each board.

The possibilities of use are almost unlimited. Therefore the use can seem somewhere relatively complex.

Anyway, the controller is preconfigured with a working state that should suit many applications. The user only needs to know a few of it to be able to start using the product. As example, for a stand-alone application, the following software will make the motor rotate at a speed of 2500PRM:

```
jog = 2500
```

To make it works, create a text file with this line, connect the motor with the cable provided by Dynamic Motion and use the software "DMComTool.exe" freely available at Dynamic Motion to upload your DM-BASIC file to the board.

Programming environment

The minimum requirement is:
A computer with a serial connexion, a text editor, The DMComTool software for downloading and a cable to connect the motor to the computer.

Dynamic Motion provides the following components:

- Notepad++ A free (GNU) text editor with special syntax coloration add-on for Dynamic Motion-BASIC language

- Free Dynamic Motion Communication Software that works under Windows XP (DMComTool.exe)

- An adaptor for the "Tinaxis" serial communication connector. For example the 9 pin RS232 (electronic board inside this cable)

- 1.8m prolongation cable

- USB – RS232 converter cable with driver CD. (needed if your computer does not have RS232 connector)

The languages

By default, 2 languages are built in:

- The software that runs inside the controller is **Dynamic Motion BASIC**.
- To remotely control the motor, a set of **DM-REMOTE commands** is available.

The register / variables name are common for both languages. The commands are different due to the needs of the language.

- The software that runs inside the board uses BASIC language for the best ease of use, portability and meaningful keywords.
- The language that is used for serial communication uses 2 letters commands for an improved bandwidth.

Dynamic Motion BASIC summary

Instructions	Expression operators	Comparison operators
IF-THEN-ELSE FOR-TO-NEXT GOSUB-RETURN GOTO PRINT PAUSE END INT (only available in BASIC 2.x) STOP START	(form: variable = expression) + - * / ^ *(power)* % *(remaining of a division)* | *(bit to bit OR)* & *(bit to bit AND)* ! *(bit to bit clear bits)* ( )	= < > <> *(not equal)* <= *(smaller or equal)* >= *(bigger or equal)*

Variables and registers	Special	Numbers
user variables: A, B, ...Y, Z *Other registers: please refer to board specific VARIABLES/REGISTER detailed description* *Example:* SPEED, IN1, ...	' *(line comment)* " *(text string descriptor)* , ; *(argument separators)*	*Line labels:* 0 to 9999999 Numbers: *signed integer in decimal notation, 32bit* *(range: from -2'147'483'648 to + 2'147'483'647)*

Remote language summary

Motion Instructions	System Instructions	Programming tools
JG *(jog, set speed)* MT *(move to, absolute position)* MY *(move by, relative move)* MD *(mode: brake, speed, ...)* DS *(Disable motor)* BR *(Brake)*	SB *(Stop Basic execution)* TB *(Start Basic execution)* RB *(RESET Basic execution)* VA *(variable change)* RV *(reset all variables to default)* SR *(return the characters "FD")*	PR *(Print the BASIC software currently in Flash memory)* PC *(Print configuration)* UL (*Upload BASIC software)*

Usage example: jg 1200enter: set the jog speed to 1200RPM

System organization

The system is organized around a central data base that contain the variables and registers.
The system continuously interact with this database base by reading the actions to perform and writing the updated values. Example: when the motor is rotating, the position counter is automatically updated each elementary time (generally 1 millisecond).

The data base can be modified by several entities. For example, the DM-BASIC and DM-REMOTE can work together to make an application that combine the functionality of an embedded software and the ability to remotely modify the movement.

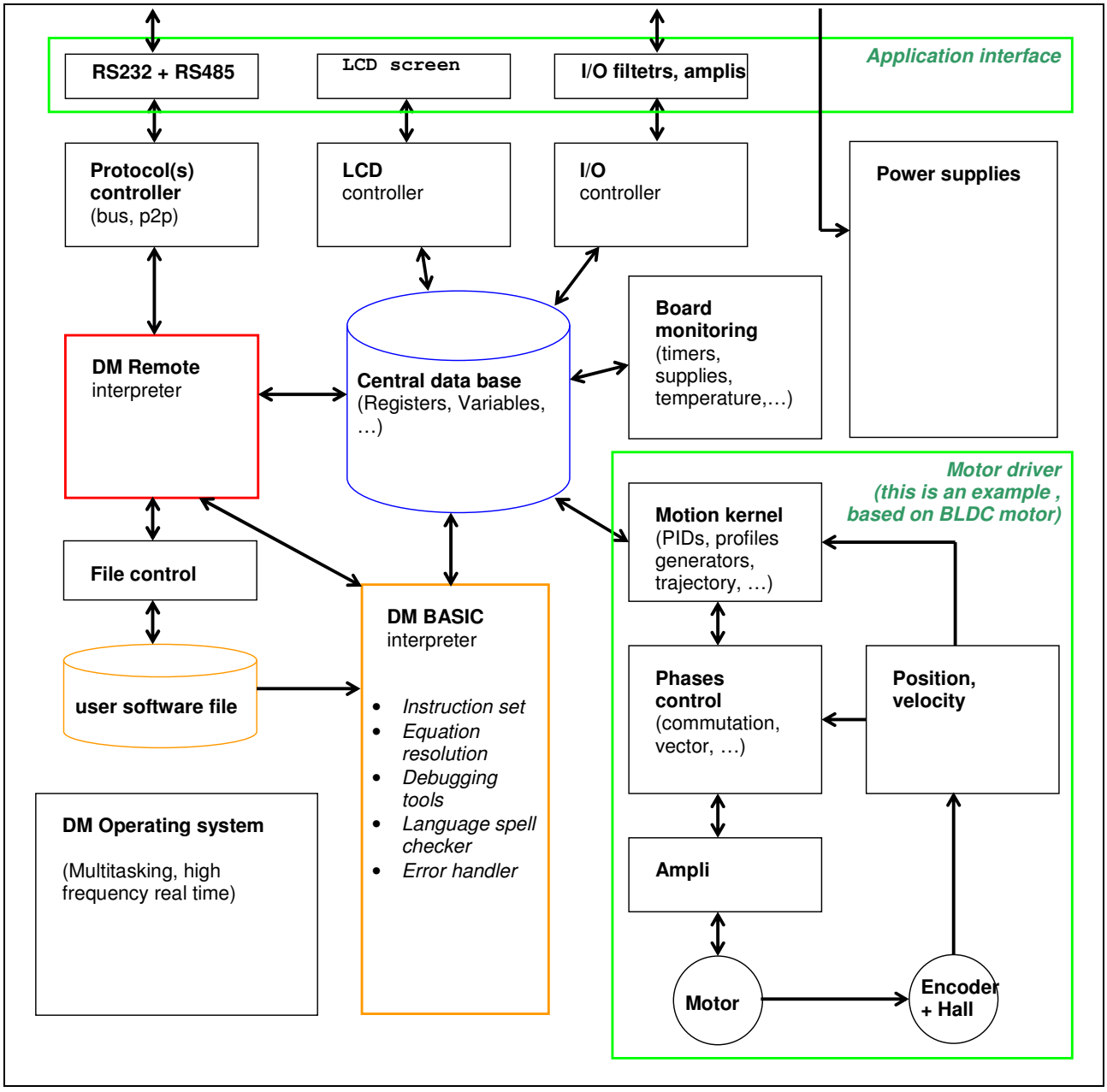


Figure 1, System organization (both software and hardware implementation)

The application software normally consists in an endless loop that read values from the database, make operations, tests and conditional operations then write to the database again.

Then the system firmware will automatically perform the actions initiated by the updated values.

At power-ON, the electronic automatically reset all values to their default state, except the flash values (named EE_x).
After reset, the serial communication is initiated to the default parameters (speed, channel, ...), and the software is started from the beginning.

The software always consists of a single text file that has been previously downloaded to the flash memory.

Getting started

Introduction

If you are not familiar with Dynamic Motion Tinaxis system, please start here, make your system working with the simplified concepts while keeping most registers unchanged. When you will be more familiar with the product, then you can go deeper in detail in the next chapter.

The principle of use is preconfigured modes that acts as wizards to configure the motion controller with common and robust parameters that should give satisfying results to most applications

Please keep in mind that the regulation algorithms, such as PID, is a complex subject. It uses many differential equations that gives a huge space for optimization and customization. Anyway, with the initial configuration, the system should already be performing in term of precision and rapidity in a satisfactory manner.

BLDC (brushless) motor controllers

The BLDC controllers have a looped control in order to control the movement.

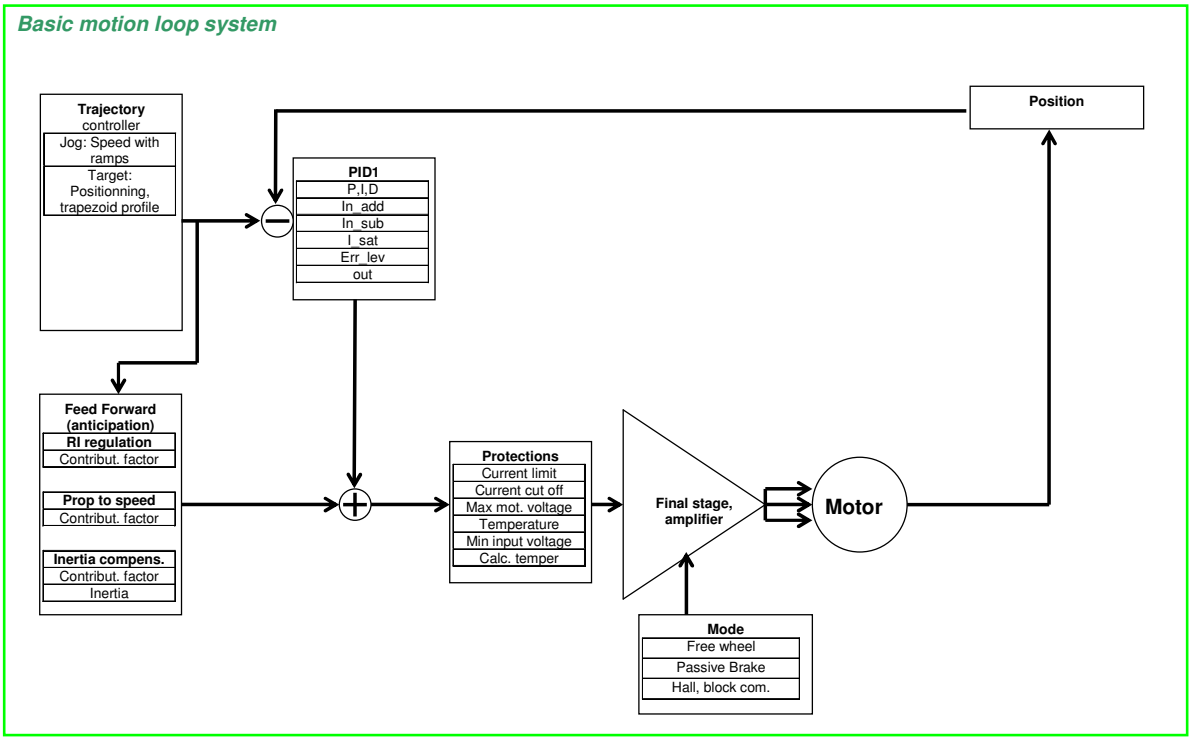


Figure 2, simple motion loop system, used in default position and velocity mode

Stepper motor controllers

The stepper motor controllers does not have the closed loop. They have independent encoder input and motor controller that the user can verify in his software.

Simple examples

Ensure you have installed the communication tool on your computer, and you have purchased 1 adaptation cable to Tinaxis connector.
(detailed instruction to install later)
Open the software "Dynamic Motion programming suite", the communication tool opens first.

Now connect the adaptation cable between the computer and the board and the power supply to the board, then switch on the power supply.

When the board is supplied, tick on "connect", automatic connection should take place. Test it by clicking on the red circle, the answer received (in red) should be like this "Motor stopped...."

Now it is ready to start designing your application
If your application require remote control, simply send commands using DM Remote language.

You can test this:
To switch ON the LED of the digital output n°1, write this direct command in the dedicated field, that will modify the register OUT1:

```
va out1 1
```

If your application require a software running inside the board, you can try this simple example

Open the Notepad++ editor that is included with the installation, save the new created file with the name you want and extension .DMB or .BAS. Assuming you name it "test.bas"

Write this tiny software:

```
10
Out1=1
Pause 300
Out1=0
Pause 300
Goto 10
```

Save it, then return on the communication software.
Select the file "test.bas" and click on upload button, and wait the end of the operation (~2 sec).

Now the software is uploaded. To start execute, you can cycle the power or click start basic.
The LED on OUT1 should blink, or an error message can be displayed if there is an error.

At this step, you have remotely took the control of the system and made your first software running.

Note that you can do both at the same time: while this software is blinking the first LED, you can remotely control the 2nd LED:

```
va led2 1
```

The BASIC language, summary

The well known language BASIC mnemonics is used. It is described in detail later in this book. As introduction, several simple examples are commented here.

Example 1 (the GOTO and PAUSE instruction)

GOTO will force the software to jump to a label that you have defined somewhere (it will not make any movement). The PAUSE will stop the software execution during a time (in milliseconds). This tiny example will blink the output, and on most boards the LED of the output.

```
10
Out1=1
Pause 300
Out1=0
Pause 300
Goto 10
```

Example 2 (the IF THEN ELSE instruction)

This is the conditional test. If the condition tested is true, then the instruction on the same line will be executed. If not, the line will not be executed and if the ELSE is present, the instruction following the ELSE will be execute.

```
10
IF TIME < 3 THEN out1=1
ELSE out1=0
Goto 10
```

This example will switch on the LED on OUT1 when the time from power-on is less than 3 seconds, then switch OFF. Cycle power to see it working.

Example 3 (the GOTSUB RETURN instruction, and PRINT)

This shows how to make a subset of software after a IF-THEN-ELSE instruction.
In this example, the PRINT is used to identify which part of the software is used.

```
10
If time < 5 then gosub 20
Else gosub 30
Goto 10

20
Out1=1
Pause 200
Out1=0
Pause 200
Print "blink  fast"
Return

30
Out1=1
Pause 800
Out1=0
Pause 800
Print "blink  slow"
return
```

Cycle power and wait more than 10 seconds to see it working. Observe the communication tool received messages...

Other examples are available within the software distribution, freely available on internet (->products->download->setup_dynamic_motion.exe). Our customer service can create new software examples on request, please use the FAQ.

Mostly used registers and variables

The registers are 32 bit signed integers. The values can swing between -2 billion to + 2billions (-2147483648 to + 2147483647 to be exact). Some of them are read-only, other are limited to a smaller range.

User variables

The 26 letters, A to Z, are user variables. It can be read and write freely. After power-up all these variables are "0".

Example of use:

```
A=10
B=2
C=A *B+10
PRINT C
```

Will show on the screen: **30**

Registers examples

This list is not complete and may differ form the actual register list present in each specific board. Please refer to "Variables/Registers detailed description" specific for the board you use.

MODE	*Typ: movement configuration wizard* *Shortcut in DM remote language: MD, BR, TM*
Predefined modes (depending on the board model, only some of these modes are available. Check "Variables/Registers detailed description" specific to the board.) the framed modes are common to all boards 0 Automatic (switch automatically between modes 4, 6 and 8) 1 Passive brake 2 Free wheel 3 Reserved 4 Voltage mode: apply a RMS voltage to the motor 5 Reserved 6 Speed regulation with loops (phase locked loop) 7 Reserved 8 Positioning (trapeze trajectory) 9 Reserved 10 Special mode to pilot 2 DC motors 11 Reserved 12 Cyclic mode (Cam profile) Depending on the board model, some modes are not implemented, and future boards will have additional modes	
JOG	*Typ: movement, speed value* *Shortcut in DM remote language: JG*

Unit: RPM (revolution per minute)
Use:
`JOG=1250`
`JOG= (IN1 * 4000)/ 10000`
Note: when MODE is 0, when jog is set, the motor will automatically switch to 6 and the motor will immediately start acceleration.
 	
Please see ACC, DEC, which are the acceleration ramps	

TARGET	*Typ: movement, position value* *Shortcut in DM remote language: MT, MY*

Unit: system increment (encoder increment when an encoder is present, or Steps of block commutation or microsteps when used.)
Use:

```
TARGET=100000
TARGET= (IN1 * 4000)/ 10000
```

Note: when MODE is 0, when TARGET is set, the motor will automatically switch to mode 8 and the motor will immediately start the move to try to reach the destination.

Please see ACC, DEC, M_SP_P, M_SP_N, which are the acceleration ramps and maximum speed

MOT_V (U_MOT in some boards)

Typ: movement, motor RMS voltage
Shortcut in DM remote language: MV

Unit: millivolt
Use:

```
MOT_V=15000
MOT_V= IN1
```

Note: when MODE is 0, when V_MOT is set, the motor will automatically switch to mode 4 and the motor will immediately ramp to the voltage
Please see V_RMP_R: the voltage ramp rate

ACC

Typ: movement

Acceleration value
Unit: RPM per second
Use: in positioning or speed modes

```
ACC=1000
```

(the motor will accelerate, from 0 to 1000 RPM during the 1st second, from 1000 to 2000 during the next one, ... until the required speed is reached.

DEC

Typ: movement

Deceleration value

Unit: RPM per second
Use: in positioning or speed modes

```
DEC=1000
```

M_SP_P

Typ: movement

Maximum Speed Positive direction

Unit: RPM
Use: in positioning mode

```
M_SP_P=1000
```

M_SP_N

Typ: movement

Maximum Speed Negative direction

Unit: RPM
Use: in positioning mode

```
M_SP_P=1000
```

This value must be positive

TIME

Typ: system, read-only

Time from power-ON
Unit: second
Use example stop the software after 1 hour:

```
If time >3600 then stop
```

TIME_U1, TIME_U2

Typ: system, read-only

Timer counting UP
Increase by 1 each millisecond when it's value is positive or 0, not counting If it's value is negative.
Use: delay an event or count a time between 2 events

Example 1: used as timer

```
Time_u1=0
10
If time_u1 < 10000 then print "waiting"
Pause 100
Goto 10
```

Example 2: used as stopwatch

```
10
if in1 < 0 then gosub 20
If time >0 then gosub 30
goto 10

if time_u1 <0 then return
print  "IN active during ",time_u1,"ms"
time_u1=-1
return

30 if time < 0 then time =0
Return
```

Note: use always a test that allows a wide time window, otherwise it is possible to miss the event because software execution is discontinuous due to processor time sharing between different tasks.

```
IF TIME_U1 =1000  THEN print "1 second"
```

This has an unpredictable behavior, must be avoid!

TIME_D1, TIME_D2	*Typ: movement*

Timer counting DOWN
Decrease by 1 each millisecond when it's value is positive, not counting If it's value is zero.
Use: delay an event or count a time between 2 events. It is similar to the timers counting UP

IN1, IN2, ...	*Typ: Input Output, read-only*

Input value. Depending on the input type, it can have theses meanings:

- Analog input, the value is the measured voltage in millivolt
- Digital input: the value is either "0" or "1"
- Time counter: the value is the time separating 2 edges, in processor cycles (if processor is 50 MHz, the time unit is 20 ns)

Use:

```
print   in1," millivolt"
```

OUT1, OUT2, ...	*Typ: Input Output*

Output value. Depending on the output type, it can have theses meanings:

- Digital output, when "0" the output is not driving current, with any other value the output is driving current.
- Special output (such as pulses generation), see de detailed description later in this book

Use:

```
OUT1 = 1
```

I_MAX	*Typ: amplifier*

Maximum current , (any modes)
Meaning: the saturation current for the amplifier, related with the measured current on the input of the final output driver.
Unit: mA
Use:

```
I_MAX=12000
```

I_MOT	*Typ: amplifier*

Measured current , (any modes)
Measured on the input of the final output driver.
Unit: mA
Use:

```
PRINT I_MOT,"milli-Ampere"
```

Note: Due to PWM energy conversion, the real motor current is generally higher. For example, with a PWM duty cycle of 50%, an input current of 1A on 24V gives a motor current of 2A at 12V.
See also I_COIL

POS, POS_HAL	*Typ: movement*

Measured position
POS is the default counter, POS_HAL is the position given by the HALL or block commutation counting in BLDC

Unit: encoder unit
Use:

Example: initialize on sensor detection

```
1 if in1 < 2000 then goto 1 'wait here while the sensor is not giving signal
POS=0
TARGET=0
```

Note: As the position is normally 1 input of the PID controller, care must be taken while changing the value or the motor can have a brutal behavior.
See also: ENC_RES

Detailed description

Registers / Variables

Please refer to the specific register list of each board for detailed list and description.

Notes about the registers index

- Each register have a unique index number
- This index may vary without prior notice, according to the firmware updates
- The user software may not use directly these numbers

Example: setting the oscilloscope channel can be done by direct addressing the register number

```
OSC_1 = 83     'THIS IS NOT RECOMMENDED!
```

Instead, always use the variable name in quote:

```
OSC_1 = "JOG"
```

- Exception is the MODBUS: through the MODBUS, it is possible to read/modify any register by it's address. It is recommended to modify only the user variables A to Z, which will never move.

BLDC controllers

This chapter applies only on boards based on DM_brushless 2.0 software and above. The boards based on DM_brushless 1.x have only 1 PID as figure 2, and are therefore not concerned by the flexible controller system.
This applies on boards:

- Tinaxis Plus BL200
- Tinaxis Plus BL201
- Tinaxis Plus BL960
- Tinaxis Plus BL120

Wizard

We have seen in the previous chapter that the MODE is a configuration wizard. How does it works?

When you set a mode, the software makes connections between individual blocks. For instance, when you set MODE=8, the following actions takes place:

- Disable the amplifier (set to free wheel mode)
- Set the trajectory generator to position profile
- Connect the trajectory output to a PID input as a reference value
- Connect the other input of the PID to the position counter output
- Connect the PID output to the amplifier input
- Connect the predictive module (feed forward) to a second input of the amplifier
- Cancel the filters inside the amplifier
- Empty the motion history and set the motion vectors to the current position and speed
- Set the amplifier in 4 quadrants mode

If the user set all these parameters 1 by 1, the result is the same. The next software will have the same result:

```
AMP_MD=0
TRJ_TYP=2
PID1_IA="TRJ_O"
PID1_IS="POS_HAL"
AMP_I1="PID1_O"
```

```
AMP_I2="FF_O"
RAMP_R=100000
PID1_M=0
SP_TAR=SPEED
AMP_MD=4
```

The figure 4 shows graphically how the connections are made between the blocks.

The Dynamic Motion products provide additional blocks that allows more complex scheme of motion loop. For instance it is possible to cascade more PID. The structure that is often used is the cascade of a speed regulation PID and a position PID.
The user has to be careful of the bandwidth of the Speed measurement. If hall sensors are used to measure the speed, the bandwidth can be too low when the motor is rotating slowly, so the regulation may be unstable at low speed. That's why this scheme is not used by default.

PID's can also be used for any other purpose, for example if an analog output is present, it can be used to regulate another process, connected to the output and measured by an analog input, totally independent form the motor, while the motor uses the other PID modules.

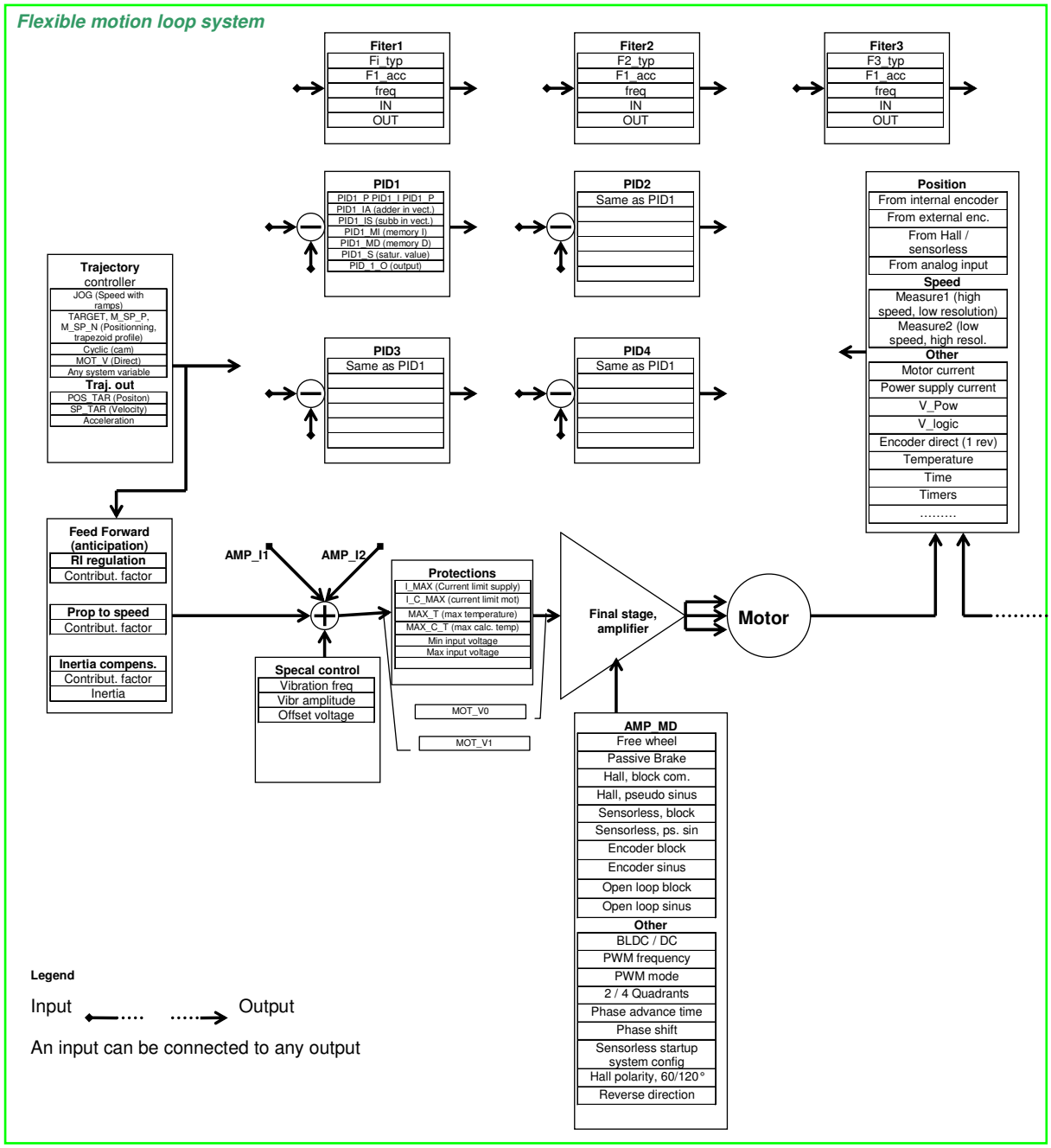


Figure 3, flexible motion loop processor and system: available blocks

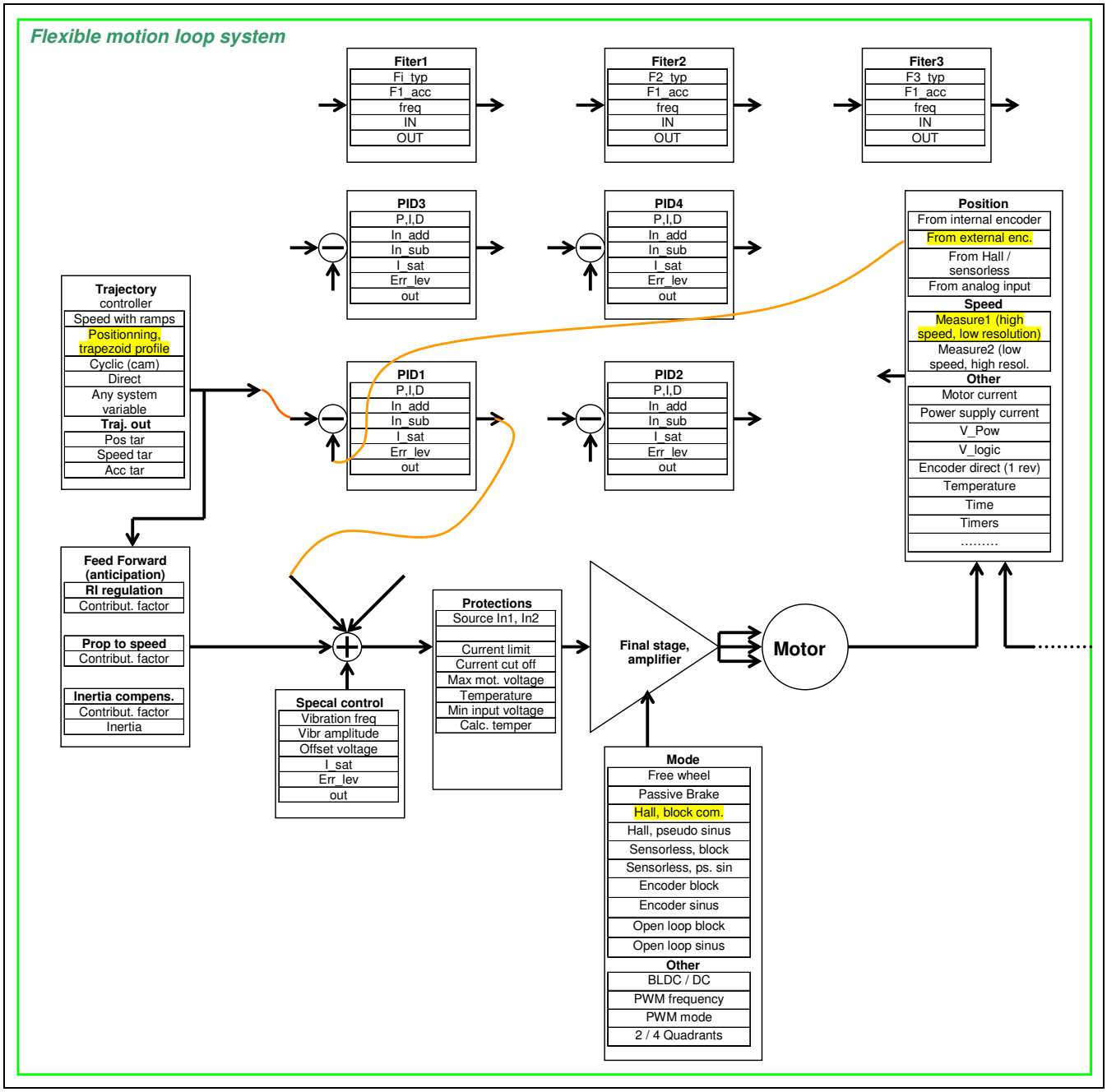


Figure 4, When MODE is changed to 8, these connexions result from that change.

The motion controller structure is created by connecting the required elements. Connexions are made by telling the name of the register connected to an input. The syntax is (example):

```
AMP_I1="PID1_O"
PID1_IA="TRJ_O"
PID1_IS="POS_HAL"
AMP_I1="PID1_O"
```

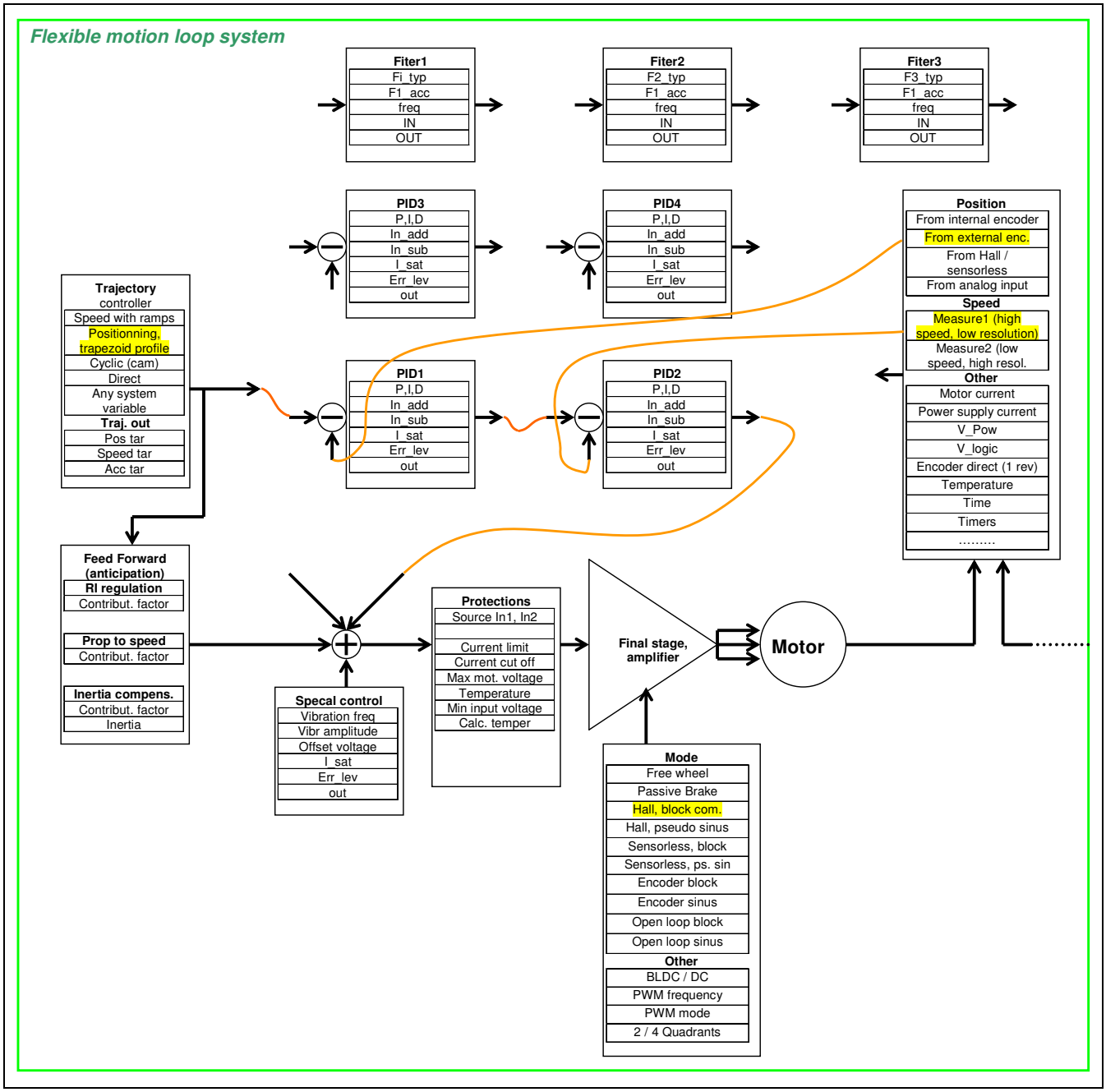


Figure 5, example of use: Connect the inputs of the needed blocks to the adequate output or any register. The oranges lines show an example of connexions for a cascade PID structure.

The "DM-basic" language

The aim of the BASIC language is to manipulate the data base that controls the motor, I/O, ...

Instruction list

	Command mnemonic	NAME	Argument	Description
1	IF	Conditional test	comparison	example IF A=0 THEN GOTO 100 ELSE A=A+1 ATTENTION: A logic expression is not implemented. Example IF A>0 AND A<10 THEN... Will not work
	THEN	imperative part of the IF-THEN conditional test	command	THEN must be followed by one instruction on the same line Multiple instruction can be done in a GOSUB RETURN function
	ELSE	Option to the IF-THEN test	command	The ELSE statement must follow an IF THEN statement, on a separate line. ELSE must be followed by one instruction on the same line
2	FOR	Loop FOR-TO-NEXT:	initialization	example FOR i=0 TO 100 PRINT "loop ",i NEXT
	TO	mark the end of loop	value	
	NEXT	Increment by 1 the variable initialized after FOR	-	
3	GOTO	Jump anywhere Must be used with many care	label	remark: do not jump in a subroutine ended by GOSUB
4	GOSUB	Jump to a subroutine	label	Branch to a sub-routine
	RETURN	Return from subroutine	-	return from GOSUB (never use alone or an error can be generated)
5	PRINT	Print a text, a value or combination of them. Print acts on embedded display or serial line, depending on configuration.	value or expression or text	exemple: locate = 0 'set curs. position print "T=";M," deg" the ";" is the TAB separator, the "," is the simple separator. At the end of the print: ";" TAB separator "," no separator before the next next PRINT "" (nothing): next print on the next line
6	PAUSE	Make a pause in the "DM-Basic" execution.	- or value	2 way of use: - without argument: wait the end of the move (in positioning mode) or the end of acceleration (in speed mode) - with argument: the argument is the time to wait in millisecond. exp 1: wait 1.5 second PAUSE 1500 exp 2: move and set the output when the move is done: TARGET = 100000 PAUSE OUT1 = 1
7	END	END of software		Not needed at the end of a file. Can be use to end the software anywhere.
8	STOP	Stop the software execution and let the movement of the motor unchanged. To restart the software, simply use the remote language word		Mainly used for debugging: when the software is stopped, it is possible to look at the variables and program line

		"tb"		
9	INT (only available in BASIC 2.x)	Declare an INTEGER variable		Use example: INT test 'declare the variable test=2+1 Print test

Operators list

	Command mnemonic	NAME	Type of use	Exemple
1	+	Addition	expression	
2	-	Subtraction Negation	expression	
3	*	Multiplication	expression	
4	/	Division	expression	25 / 10 result=2
5	%	Remaining of a division	expression	25 % 10 result=5
	^	Power	expression	10^3 result= 1000 49^-2 result=7 (square root) (note: ^-3 and above are not implemented)
	&	bit to bit logical AND (only bit to bit operation, no logical expression)	expression	7 & 9 result 1 7 binary = 0111 9 binary = 1001 binary result: 0001
	|	bit to bit logical OR (only bit to bit operation, no logical expression)	expression	7 | 9 result 15 7 binary = 0111 9 binary = 1001 binary result: 1111
	!	bit to bit clear bits (only bit to bit operation, no logical expression)	expression	7 ! 2 7 binary = 0111 2 binary = 0010 binary result: 0101
	(	open brace (used in math expression only)	expression	Attention: limit at 10 nested braces or lower, depending on model
	)	close brace	expression	
	=	equal	assign a variable, comparison	A=0 IF A=0 THEN GOTO 100
	>	larger	comparison	if A > 0 then goto 10
	<	smaller	comparison	
	<>	not equal	comparison	
	>=	larger or equal	comparison	
	<=	smaller or equal	comparison	

Special

	Command mnemonic	NAME	Use	Example
1	'	inline comment	The comment start with ' and ends ant the end of line	A=0 'initialize A
2	"	Quote	input a text string get the number of a variable	print "hello" osc_1 = "pos"
3	number	LABEL	Define a point in the program to jump to. Max 7 digits (or 3 digits with low memory models)	100 A=A+1 GOTO 100
4	: (only available in BASIC 2.x)	LABEL	Same as before, for free name	yellow: If in1=0 then goto yellow

5	charact. H00	end of file	If you download the software with a custom tool, the character H00 indicate the end of file	
6	,		argument separator	
7	;		in "print" command, separate by tab.	

Other rules

- Each line must be below or equal to 80 characters (ore less depending on memory option)
- Any number of space or tab can be used to separate elements
- One line can be build of
  - one label (number)
  - one or more conditional test
  - one instruction
  - comments
- It is not possible to but 2 instructions on the same line, except "IF THEN"
- The commands and variables can be written uppercase or lowercase or a mix of them.
- The numbers are 32 bit signed integer. Each result is rounded by truncation to lower value (ex. 1.8 is truncated to 1, -3.5 truncated to -4). A 32 bit value range is:
  -2147483648 to + 2147483647
- The software is not compiled. That means that if an error is present, it will be discovered when the software reach it. In some cases, the software will take long to reach it, for example the following error will be discover after 1 hour of working:

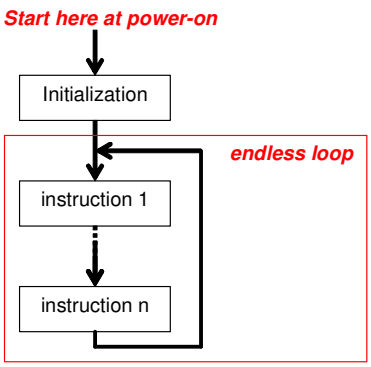
```
'main software
10                                      'infinite loop
      TARGET = (TIME * 1024)/60         'The rotor move like the watch needle
      IF TIME > 3600 THEN GOSUB 100     'routine at label 100 that does
                                      'not exist, will generate error
GOTO 10

'sub-routine
100   PRIN#ç&T "1 hour "   'there is an error in the PRINT instruction
      RETURN
```

- RS485 and the BASIC "PRINT" command: The Print command send the stream to the destination (LCD and/or Serial communication), regardless the transmission already occurring in the serial communication. When using the programming cable (duplex), there is no problem. When using the RS485 connection (half duplex), it may create collision if an input communication is present at the same time. Our advice is to not use the Print command together with RS485 (UART set to 0, 1, 4 or 5) or (PR_CONF set to 1).

Programming tips

- To make software that gives the illusion of a continuous treatment, an endless loop can be used. For example, a loop that sequentially measures an input voltage, print the speed on LCD and set the JOG speed proportional to the input. (chart on the left)
- Make structured software and limit the use of "GOTO". Use GOSUB-RETURN instead.
- Use line indent to clearly see where are the grouped instructions
- Use comments, but not too much to save memory



space and execution speed

```
'main software
GOSUB 100                                  'initialization is at label 100
10                                         'endless loop
      GOSUB 200                            'routine at label 200
      IF IN1>2000 THEN GOSUB 300    'routine at label 300
      ELSE GOSUB 400
GOTO 10
'sub-routines
100 ….
200 ….
300 ….
400 ….
```

Dynamic Motion REMOTE language reference

The remote language can control the BASIC software execution, and also manipulate the Data base. It is then useful for developing the application, tests and also can be used to remotely control the movement.

Structure of an ASCII command:

examples:
simple command, without argument
br enter to stop the motor supply and brake

command with numerical argument
jg space -1000 enter set the motor to run at -1000RPM
(a positive number is without the +)

command with text argument and numerical argument
va space PID_P space 45 enter set the PID_P variable to the value 45

(space or enter is the character "space" or "enter")

Redundancy check

In order to increase the safety of the transmission, the redundancy check is a simple duplication of the message:
Example:
A message of the form jg space -1000 enter
becomes jg space -1000 space jg space -1000 enter
This form is absolutely not optimized in term of rapidity of transmission, but is very simple to implement on any platform used as master, including PC and PLC (Programmable Logic Controller).
For a speed optimized protocol, please contact the manufacturer.

Bus addressing

When the variable "BUS_ADR" has been set, the language must be adapted: every command must start with the address number in 3 digits. An address "000" is always accepted.
Example:
If the address has been setup to 234 , the message jg space -1000 enter
becomes 234 jg space -1000 enter
or with use of redundancy check:
234 jg space -1000 space 234 jg space -1000 enter

Commands list

	Command mnemonic	NAME	Argument	Description
1	tb	START BASIC	no	
2	sb	STOP BASIC	no	
3	rb	RESET BASIC	no	Restart BASIC from beginning. It does not reset all the variables to default
4	ul	UP_LOAD	file	Upload the basic file. Use the character 'H1C (28) to end the transmition (automatically done by the DMComTool)
5	pr	PRINT SOFTWARE	no	Print the BASIC software on the serial port
6	ds	DISABLE MOTOR	no	Make the outputs in "open" state (set MODE to 0)
7	br	INSTANT BRAKE	no	Short the motor phases to ground (set MODE to 1 and stop BASIC)
8	md	CHANGE MODE	number	Change the motion mode (see BASIC variable)
9	jg	JOG	number	set speed in speed modes (set the value to JOG)
10	my	MOVE BY	number	Relative move, in position mode (execute TARGET = TARGET + value)
11	mt	MOVE TO	number	Absolute move, in position mode (execute TARGET = value)
12	va	CHANGE VARIABLE	variable + (number)	Change a variable (see variables table in the Data Base section) if the number arg. is omitted, it prints the actual value on SERIAL port
13	pc	PRINT CONFIG	no	Print all the variables on Serial port
14	pl	PRINT ON LCD	char. string	Direct print on LCD (for test purpose)
15	rv	RESET ALL VARIABLES	no	Reset all variables to their default values
16	sr	Search motor	no	Returns "fd". Useful to check if the connection is working
17	dl	Download Oscilloscope datas	no	When this function is available, please refer to the "oscilloscope function" description
18	tm	Test move	no	Makes the connected motor to move slowly

Notes

1. The ASCII character "space" is 'H20 (hexadecimal 20 = decimal 32), the character "enter" is 'H0D or 'H0D + 'H0A
2. Commands can be sent in capital or small letter or a mix of them. If redundancy is used, the same type must be used twice.
3. The number of spaces can be 1 or more. The same comment as #2 is also valid about the redundancy.

MODBUS

Modbus RTU on RS485 is optionally available on some boards

Use:
Set the MODBUS address of the motor

BUS_ADR = 10

Select speed, between 9600, 19200, 34800, 56700 bps

SER_SP = 19200

Select parity bit, stop remote communication and start modbus

UART= 101

(100= MODBUS with no parity, 101= MODBUS with ODD parity, 102= MODBUS with EVEN parity)

Address mapping
All the variables are mirrored at adr 512 to ~700, converted to signed 16 bits by truncation, and also mirrored at address 1024 to ~1400, in signed 32 bits. 32 bit access is possible by accessing the lower half of any 32 bits value in the base address and the most significant bytes at the next address. The variables number varies upon board model, therefore use the remote command pc enter to get the variable list with their index code.

A special mapping is possible on request.

Example:
a= 1000
c= 100000
POS=1000 * 1024 '(1000 motor revolutions)

The variable "a" has the index 0
The variable "c" has the index 2
The variable "POS" has the index 26

modbus value at address 512: 1000 (read value of a)
modbus value at address 514: 32767 (Hexa 7FFF) (read truncated value of c)
modbus value at address 538 (512 + 26): 32767 (Hexa 7FFF) (read POS)

modbus value at address 1024: 1000
modbus value at address 1025: 0
modbus value at address 1028: 34464 (Hexa 86A0)
modbus value at address 1029: 1 (Hexa 1)
modbus value at address 1076 (1024 + 2*26): 40960 (Hexa A000)
modbus value at address 1077: 15 (Hexa F)

Check: if we recalculate the numbers in 32 bits:
$34464 + 2^{16} * 1 = 34464 + 65536 = 100000$
$40960 + 2^{16} * 15 = 40960 + 65536 * 15 = 1024000 = 1000$ motor revolutions

MODBUS specifications
The MODBUS implementation is limited to MODBUS RTU, with 5 commands:

- Read register (n° 4)
- Read multiple registers (n°3)
- Write single register (n°6)
- Write multiple registers (n°16)
- Read / write multiple registers (n°23)

Please refer to the norm for further specifications. It is available on internet at:
http://www.modbus.org/

Notes
When RS485 is used, the remote cable must be physically disconnected, because both share the same COM port inside the driver board. The remote cable has priority and would block communication. Note that it is possible to use MODBUS on the RS232 cable.

The downloading tool: DMComTool.exe

The software is freely available from Dynamic Motion, simply login and go to PRODUCT DOWNLOAD menu. If you are facing troubles send us an email to "welcome@dynamicmotion.ch" and we will e-mail you the link to download the software.

The Dynamic Motion Communication tool works under the ".NET" environment freely available form Microsoft.

Installation

The software is provided as a single installation file to download. The Install wizard will automatically create a directory on your computer with documentation, examples and software.
If the ".NET" environment is not present on your computer, please go to the Microsoft Internet page
http://www.microsoft.com/downloads
Then look for the .NET framework (version 2.0 or later) and follow the installation instructions.

Instruction of use

General

This software controls the communication with the motor. The main window shows every messages that are communicated.
Blue: messages send from the computer to the Dynamic Motion controller board
Red: messages sent by the board to the computer

Send the software you have made

1) Select on which COM port is connected your cable (1). When you use an USB adaptor, it's driver creates an additional COM port on your computer (for example COM5). If the motor is connected, the "AUTO" mode will automatically find the right port.
2) Open the connection (tick the "connect" checkbox) (2)
3) Select the software you have made, that you want to send (upload) to the driver (6) and (3)
4) Click on "Send" icon (7) when your software is selected.
NOTE: Your software must be edited in a separate editor. Do not forget to save your file in the editor before sending it to the driver.

Verify that everything is working well

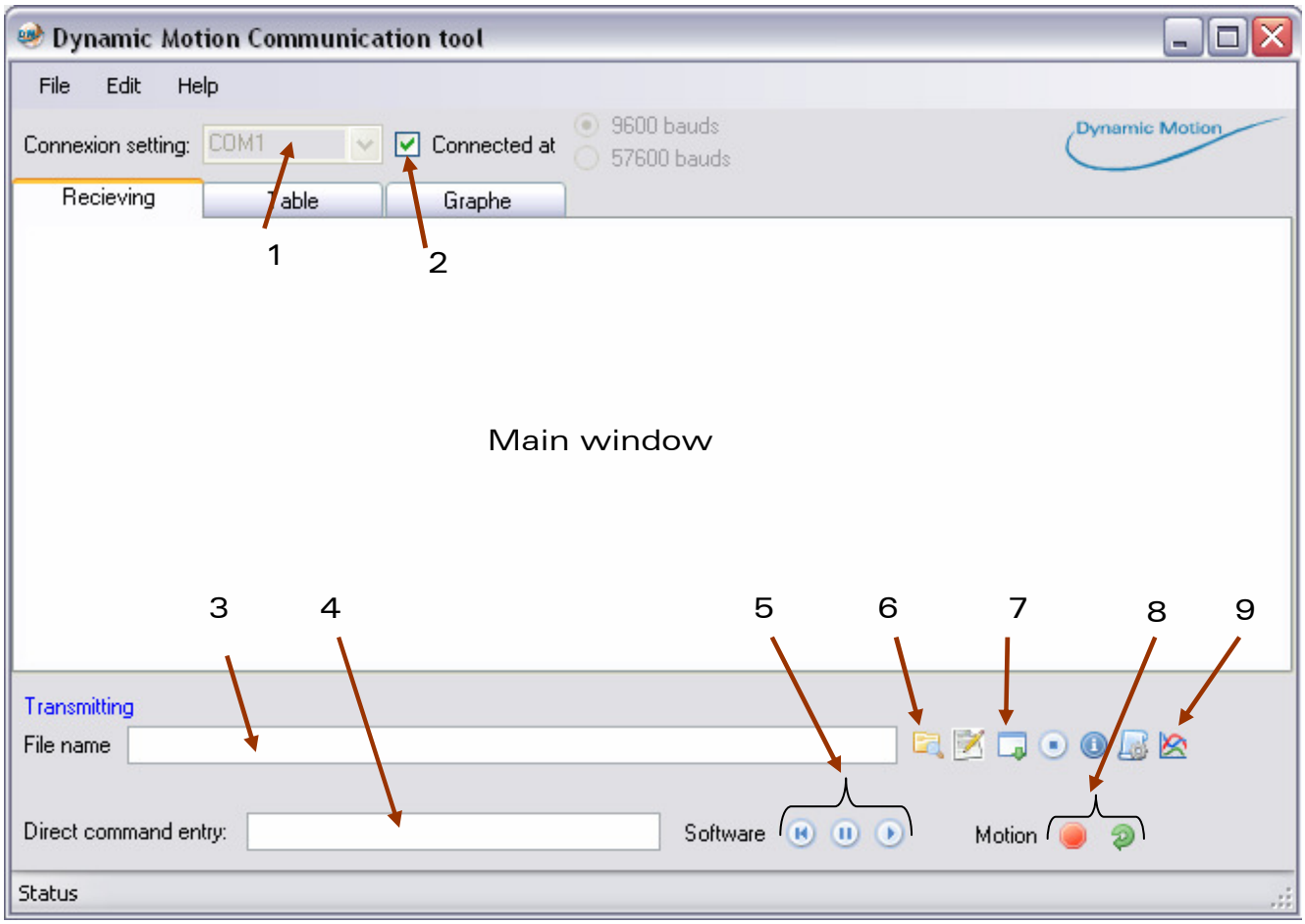
Use the buttons "move" and "stop" to check if everything is OK (connections, ...)
You should see blue and red messages on the main window. If you do not have red messages, refer to the troubleshooting chapter

Debug your software

- Use the "print" instruction in your basic file, and get information in the main window
- use the "play", "pause", "restart" (5) to manage the software execution

Save the software present inside the board to your computer

First click on to download the software. At the end of transmission, use the menu "File" "Save".



Troubleshooting communication between computer and board

- If there is no communication (no red messages on the main window when you click on "stop" or "move" (8)):
  - Check that the board is powered on its "logic supply input" or "power supply input", depending on board model. Almost all our products have a LED showing that logical power is present
  - Check the RS232 cable connection, and the correct USB installation if you use an adapter
  - Check that you have selected the correct com port (1)
- If there is communication but the motor does not move
  - If the board has separate power inputs for logic and power, verify that the "power supply input" is connected
  - Click on and look at some returned values, especially "mode", "v_in", "u_mot"
  - In BLDC motors, if an HALL sensor is damaged or disconnected, it will not work. Please look at our BASIC examples for a software that test the HALL inputs
- If you do not use Windows XP based computer, we do not provide the software. It may work on Vista and some other platforms but we did not test it and we don't provide support.
- If you need to work WITHOUT our tools you need:
  - a terminal software that will be used for communication
  - a text editor software. Notepad ++ can be used on other platforms, please refer to http://notepad-plus.sourceforge.net. Then manually replace the syntax coloration file by the one we provide
  - use the DM-Remote language to control the electronic board

- If the communication software starts normally but hang when trying to connect to the com port: The most probable reason is that there is a problem with the installation of the com port hardware. How to check and correct it:
  - Go to the control panel (part of Microsoft Windows), system, material, peripheral manager, and then find the com port you want to use. Check and correct the driver from this section.
  - To check if communication with the PC works, use a Null Modem serial cable (twisted cable) and connect 2 different com ports together. Use 2 instances of any terminal software or Dynamic Motion communication software, and check if communication is going from 1 to the other, and vice versa.
- If the com port seems not to work properly, please check it with a loop between outgoing and incoming messages: plug a loop connector on your computer com port (pin 2 and 3 connected together) and check if you get the same messages between outgoing and incoming (red and blue with in the dynamic motion communication tool). Check the difference if you disconnect the loop connector. Example of loop connector: pin 2 and 3 together, with a female 9 pin D-Dub connector.

The oscilloscope function

Some motor versions include a feature of recording to visualize parameters evolution. This function is called oscilloscope, it can supervise two variables of your choice, with flexible sampling time between 1 millisecond and 500 hours.
The most common use of this function is the movement check, by showing the tracking error: "MOV_TAR" and "POS".
Long time recording are possible, for example to measure the temperature within 24 hours.

Instruction of use

1. Select the variables that you want to record

Assign the variable number to the oscilloscope channels **osc_1** and **osc_2**
For example: showing the real position and the instruction position

To do this in BASIC:

Write the following lines in your software, upload and execute it

```
osc_1=”pos”                 '(to show real position)
osc_2=”mov_tar”   '(to show instruction position)
```

or also:

```
osc_1=26     '(26 is the index for "POS" variable)
osc_2=44     '(44 is the index for "MOV_TAR" variable)
```

To do this with remote language:

```
va osc_1 "pos"
va osc_2 44
```

(to get the variable index, get the configuration by clicking on icon)
Warning: the variable number can change according the electronic version and options

2. Set the sampling time and start measuring

By setting the sampling time, the recording immediately starts by saving the first point.
To do this in BASIC:

```
osc_t=3              '(record each 3 ms)
```

To do this with remote language:

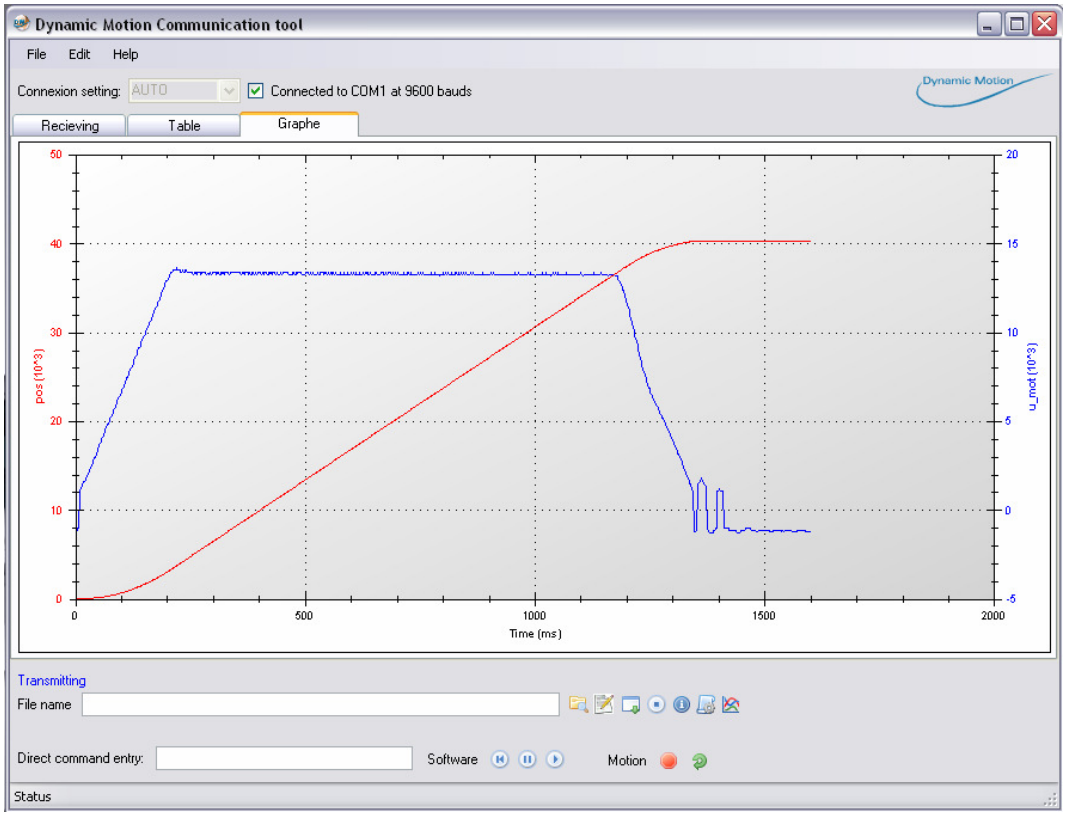
va osc_t 3

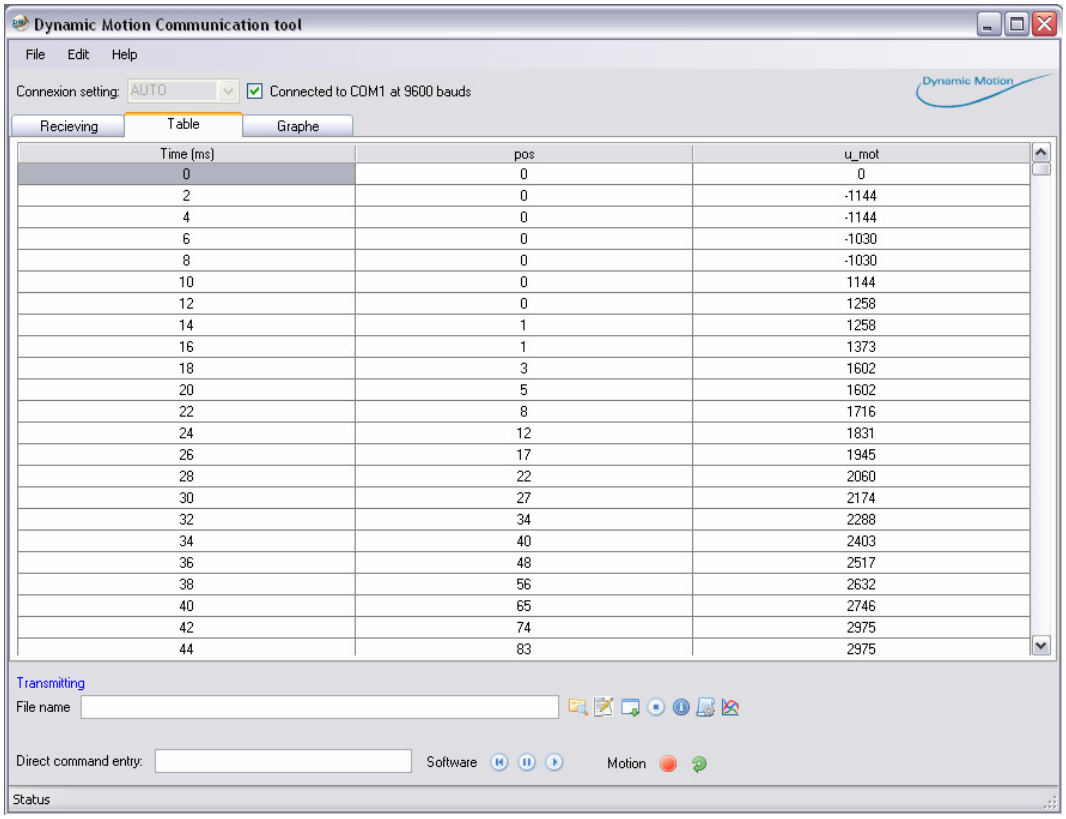
3. Get the graph and table of values

Wait enough time to record the values that you want to analyze, then send the request to the board: click on the appropriate icon (9) or send “dl” via the direct command field (4).
Be patient, the download can takes up to 20 seconds.

Graph and table Tab

The Dynamic Motion communication tool has two tabs which are reserved for oscilloscope function. One is a table composed of three columns (Time, channel 1 and channel 2) and as rows as number of points, and the second is a graph which shows data contained in the table.





When the table and the graph appears, it can be saved in the chosen folder or printed (From the "File" menu).

Graph option

Zoom is available by left mouse click and mouse wheel. Plan by middle mouse click. Zoom is centered on the mouse cursor.
Right click shows a context menu with 5 function:

- Copy: To copy the graph to clipboard as bitmap image
- Equalize Y scale: To have a same Y scale for the two variables
- Show point values: To show the value of each points of each variables
- Un-zoom or Un-pan: To cancel the last zoom or pan
- Undo all zoom/pan: To cancel all zoom and pan
- Show zero: If you need to view the zero with auto zoom

Notes

<u>How to know observation time:</u> Each motor have a different memory depth available for this function. So, to know how many points are available, you should use the direct command field (4) and send:

- va osc_p (It's read only variable)

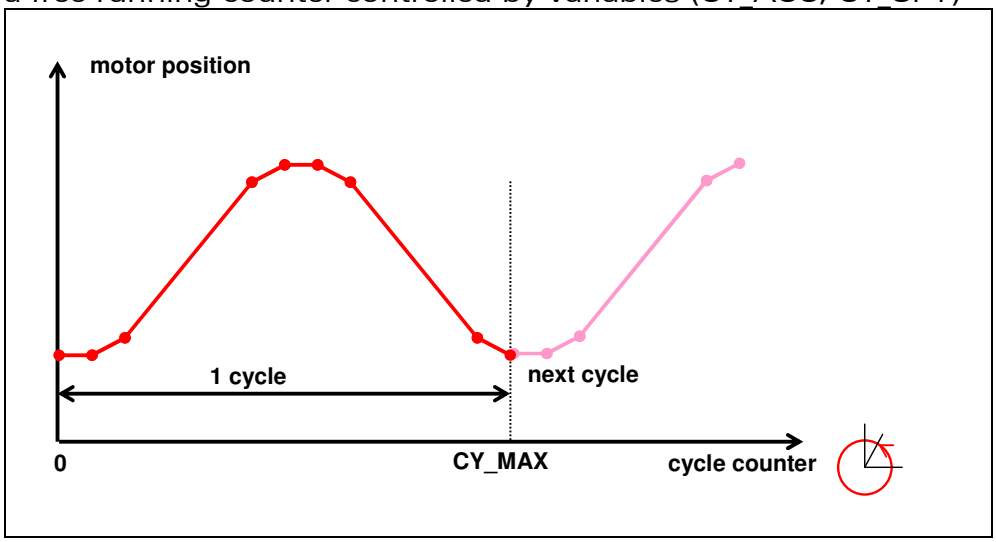
With this notice, it's possible to define observation time with sample time defined with this equation

$$\frac{Observation\ time}{osc\_p} = osc\_t$$

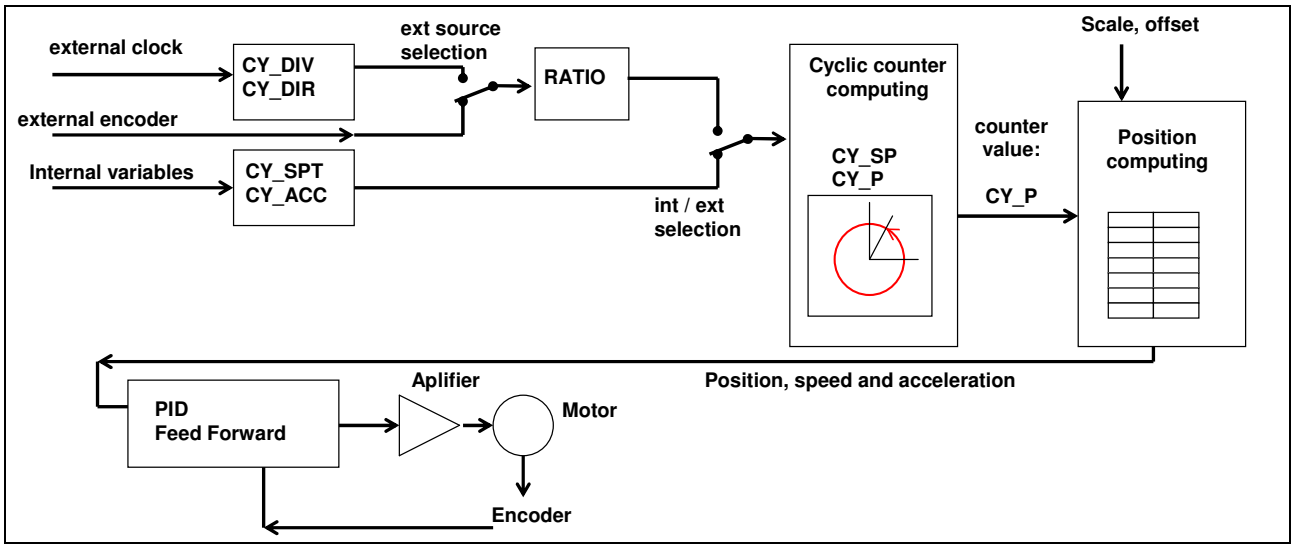
The graph is always shown with x axis as time. For further analysis, it is possible to save and export the data to any software such as OpenOffice Calc or Microsoft Excel. Example of use: Show the result with a logarithmic scale, or use both data set as x and y values (ex.: show the graph of U_mot =f (speed)).

Cyclic movement

Optionally available on some boards
This mode has the aim to make the electronic equivalent to a mechanical cam. The cam axis is represented by the cycle counter, that counts at a variable speed.
The speed of this counter can be synchronized with external events (encoder, pulses) or a free running counter controlled by variables (CY_ACC, CY_SPT)

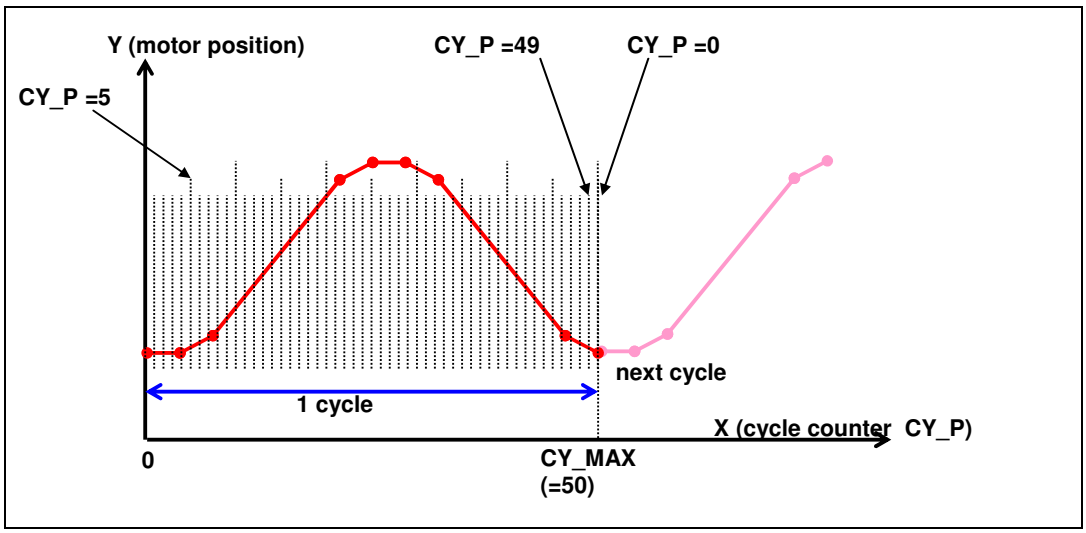


The rotor position is controlled with a table of points, that makes the correspondence between the cycle counter and rotor position (represented by the red dots on the curve).
Configuration
The synoptic below represent the information path. A key element is the table of points. The rotor position for any cyclic value is computed from the lines connecting the points.



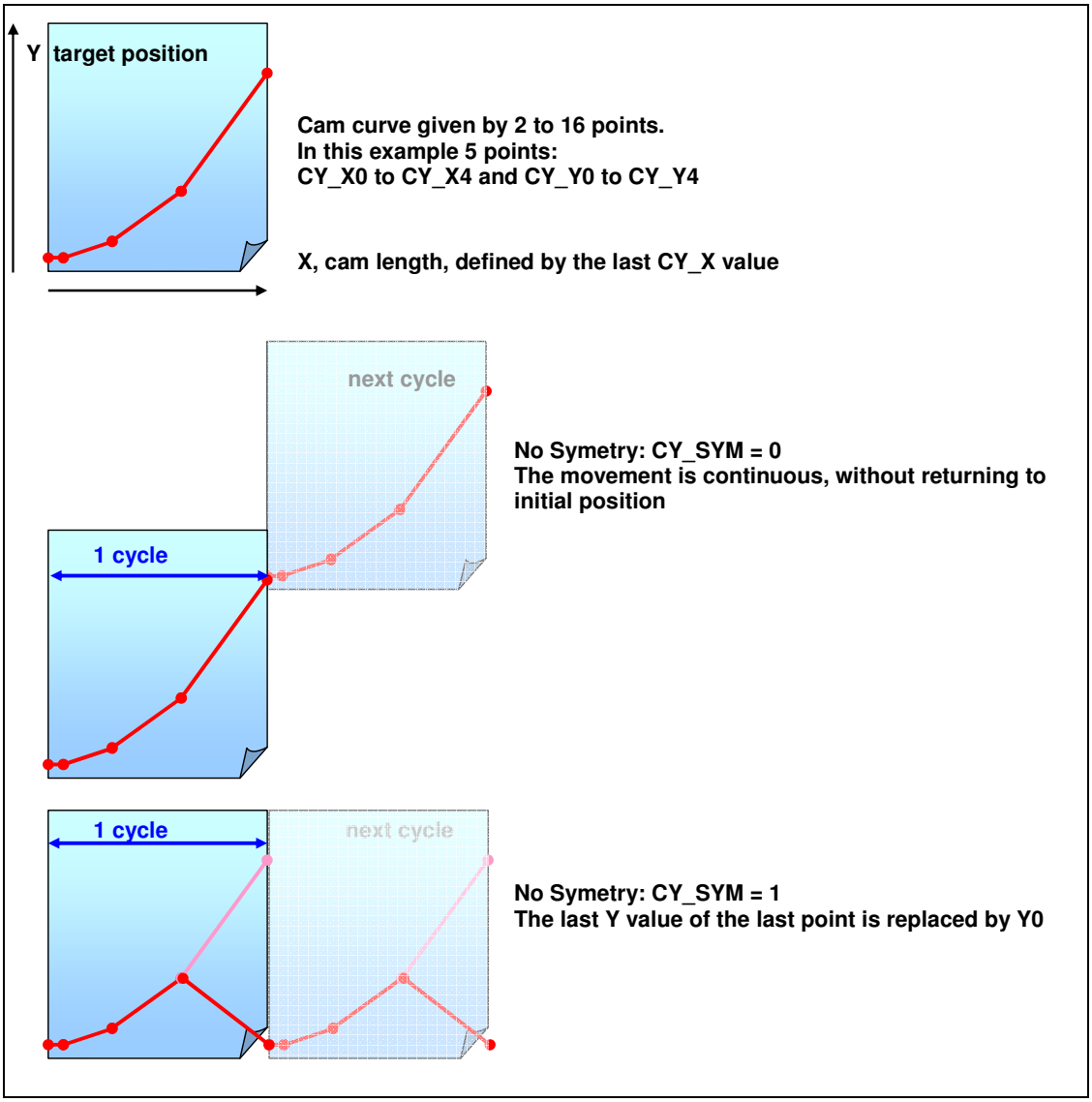
Detailed operation

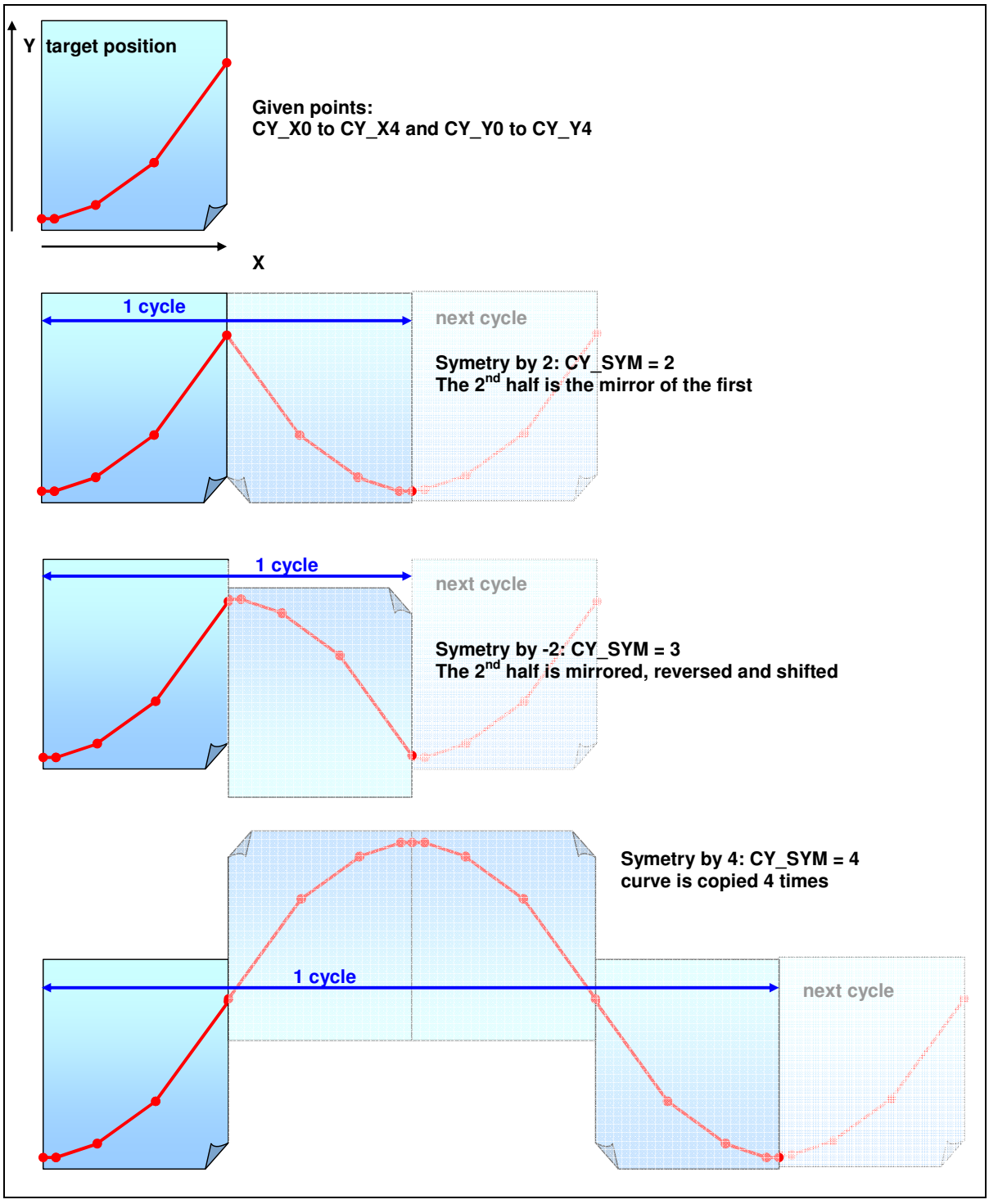
The cam curve is given by a few amount of points. Then the segments between the points are internally used to calculate where should be the rotor position at any X value. The X axis is named CY_P. The X value can be compared with the angle of a virtual cam. 1 full revolution of the virtual cam is done in CY_MAX units, that means that is depends on the table of points given by the user. The following illustration shows a CY_MAX value of 50, but in reality we recommend values between 1000 and 100'000 to have an optimal resolution.



Symmetries, principle of repetition

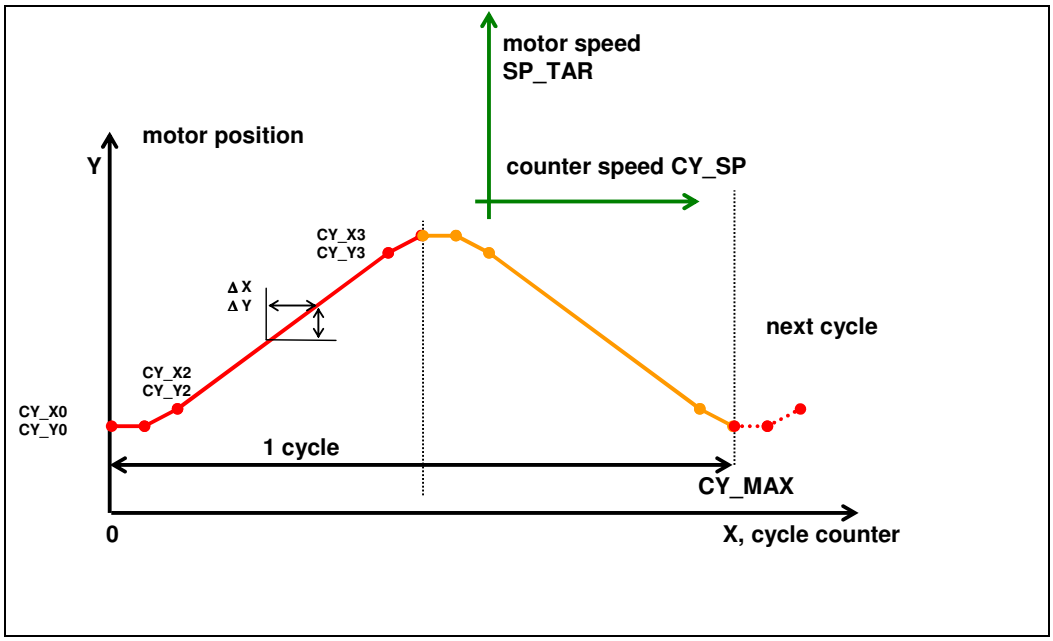
The system offers 5 principles of repetition. The repetition period is named 1 cycle. Depending on the repetition principle, the cycle can be the length of the maximum X value, or a multiple. The next figures illustrate the 5 available possibilities.


Y
target position
Cam curve given by 2 to 16 points.
In this example 5 points:
CY_X0 to CY_X4 and CY_Y0 to CY_Y4
X, cam length, defined by the last CY_X value
next cycle
1 cycle
No Symetry: CY_SYM = 0
The movement is continuous, without returning to initial position
1 cycle
next cycle
No Symetry: CY_SYM = 1
The last Y value of the last point is replaced by Y0



Tips and tricks (cyclic movement)

- Calculate the motor speed



The motor speed is given by the counter speed (CY_SP or CY_SPT) and the slope.

Example: the slope of the segment 2-3 is: $slope = \frac{vertical\_distance}{horizontal\_distance} = \frac{CY\_Y3 - CY\_Y2}{CY\_X3 - CY\_X2}$

The scale factor changes the slope. With a scale factor of 1024, the factor is 1 (100%).

$$slope = \frac{vertical\_distance}{horizontal\_distance} \cdot scale = \frac{CY\_Y3 - CY\_Y2}{CY\_X3 - CY\_X2} \cdot \frac{CY\_SCL}{1024}$$

$$motor\_speed = slope \cdot CY\_SP = \frac{CY\_Y3 - CY\_Y2}{CY\_X3 - CY\_X2} \cdot \frac{CY\_SCL}{1024} \cdot CY\_SP$$

The unit of CY_SP is the number of X elements per second. Therefore the unit of the calculated motor speed will be the number of encoder elements (or microsteps) per second.

Numerical values:
Let's imagine a case
CY_X2=2000
CY_X3=8000
CY_Y2=1200
CY_Y3=3800
CY_SCL=512 ' scale factor of 0.5
CY_SPT (=CY_SP)= 30000

$$motor\_speed = slope \cdot CY\_SP = \frac{3800 - 1200}{8000 - 2000} \cdot \frac{512}{1024} \cdot 30000 = 6500[enc\_pt/\sec]$$

In RPM if the encoder resolution is 1024:
6.3477 revolution per second = 380 RPM

Cyclic movement variables, complementary information

CY_SPT	Cyclic movement: cam speed target	RW	Unit: curve elements per second
CY_P	Actual position (=X axis)	RW	Unit: curve elements The value is between 0 and CY_MAX
CY_ACC	Acceleration When CY_MOD=0 (internal cyclic counter calculation), this value is the acceleration / deceleration that link the speed target	RW	Unit: speed evolution each millisecond

	(CY_SPT) and the speed (CY_SP)		
CY_MOD	Cyclic mode, input selection	RW*	-1: Cyclic movement calculation, but no motor move. Example of use: calculate the CY_TAR value and use it somewhere else later. cy_mod=-1 mode=12 pause 5 mode=8 target=cy_tar 0: internal (CY_P calculated from the speed CY_SP) 1: encoder input 2: pulses input, counting on rising edge
CY_S_D	Input scale divider	RW	scale factor between the encoder / pulses inputs and the y_sp Note: the CY_P can be updated more often than the inputs (if multiplier is bigger than divider) or less often in the opposite case. The calculated speed between the last events is used.
CY_S_M	Input scale multiplier	RW	scale factor between the encoder / pulses inputs and the y_sp
CY_NM	calculated last point of the cam. Does not include symetry	R	Unit: points min 2 valid points, max 16
CY_MAX	Last calculated element of the cam, including symmetry. Calculated from the last valid X value.	R	
CY_NO	Actual point used in the cam calculation	R	0 to 16
CY_X0	X value of the initial point of the curve	R	always 0
CY_X1 to CY_X15	X coordinate of curve points	RW*	
CY_Y0	Initial Y value of the curve	RW*	
CY_Y1 to CY_Y15	Y coordinate of curve points	RW*	
CY_SYM	Symetry selection	RW*	
CY_SP	Actual speed of the X axis	R	unit: element / sec
CY_SCL	Y axis scale. 100% = 1024	RW	
CY_OFS	Y offset of the slope	RW	
CY_TAR	Motor position (Y) target Value used as input of the PID when the cyclic mode is in use.	R	This value is valid 2 milliseconds after first start of MODE 12 and stay valid when leaving MODE 12.
CY_CNT	Count the number of cyclic revolution. 1 full period increment/decrement by 1 when the CY_P reach 0 by a positive/negative value of CY_SP.	RW	

R= read only
RW= read and write
RW* = the value is RW when the cyclic mode is not activated, and read only when the mode is started.

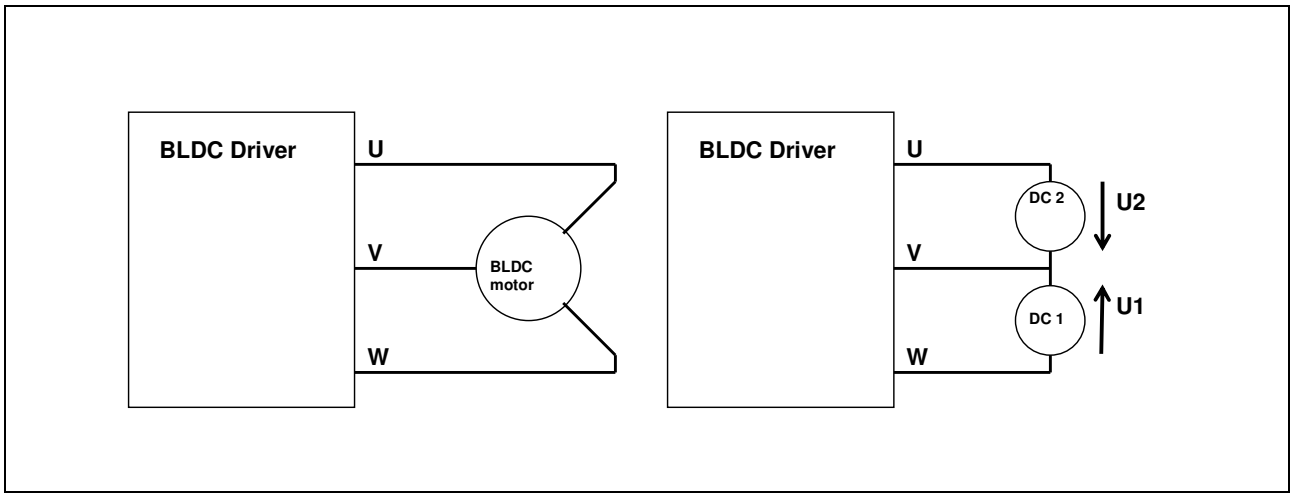
When the command "MODE=12" is used, the complete cam is checked and calculated.
The following conditions MUST be respected:

- The valid X values are different from each other
- The X values are in increasing order
- The first X value that does not respect these laws mark the end of the table, and the previous point is used as the end of the slope and X size of the cam.
- There is at least 2 valid points

If an illegal condition is detected, the mode will not change to 12.

Use a DC motor with BLDC boards

In BLDC controllers, it is possible to select the mode 10, that allows the connection of 1 or 2 DC motors.
The limitation compared to normal DC driver board is the middle point between the 2 motors. The allowed voltage of U, V, W is 0 to V+ power.



Let's take some examples:
U0 is the input voltage (V+power- 0V)

$U_1=U_0$, $U_2=U_0$: possible
$U_1=-U_0$, $U_2=-U_0$: Possible
$U_1=-U_0$ and $U_2=U_0$: not possible. In this situation, $U_1=-½U_0$ and $U_2=½U_0$
$U_1=U_0$ and $U_2=-U_0$: not possible. In this situation, $U_1=½U_0$ and $U_2=-½U_0$

The limitation to ½ U_0 is done automatically.

How to tell by software what are the voltages?

AMP_I1 and AMP_I2 are used to point to the registers used for each voltages. For example, set them to the output of PID1_O and PID2_O:
AMP_I1=" PID1_O"
AMP_I2=" PID2_O"

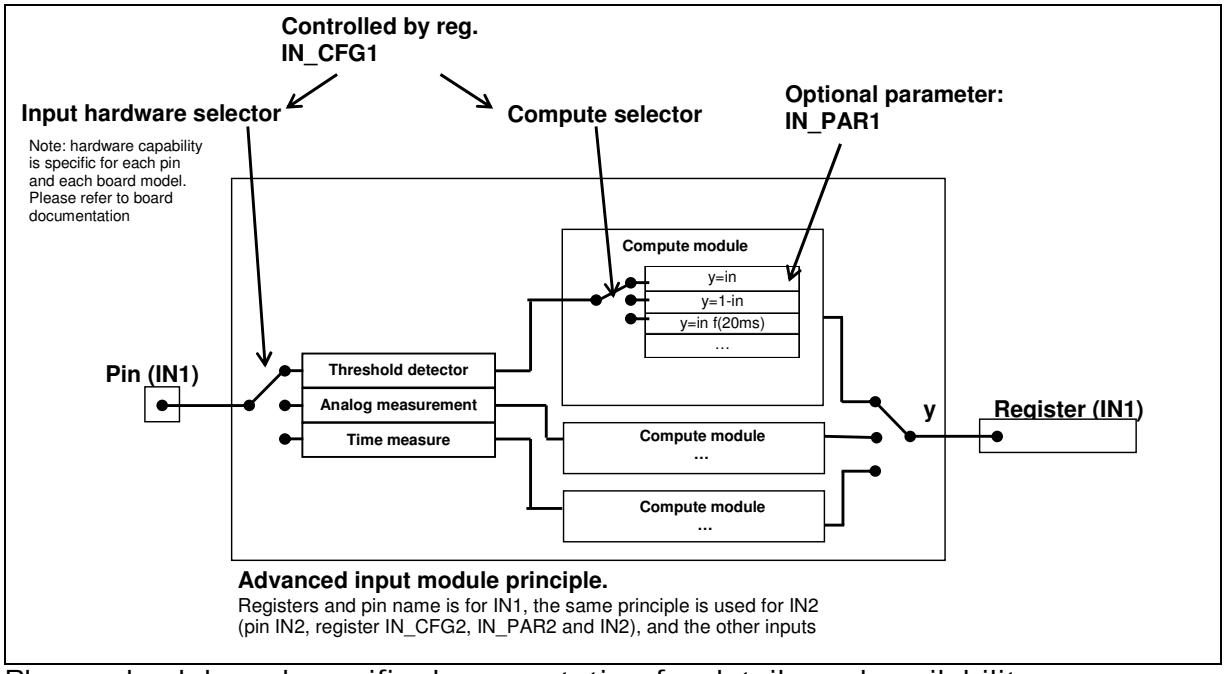
Other example: DC motor associated with an encoder. There is no wizard such as "MODE 10" that complete the configuration. You can do it manually:
MODE = 10 'base configuration that is close to the result
PID1_IA= "POS_TAR"
PID1_IS = "POS_XEN"
ENC_RES=4096
TRJ_RES= ENC_RES
AMP_I1= "PID_O1"
TRJ_TYP=1 '1 for speed mode, and 2 for positioning mode

Advanced Inputs / Outputs features

Depending on the board model, special inputs and outputs features are available

Inputs
Related registers:
IN1, IN2, IN3, ...: registers that contains the result

IN_CFG1, IN_CFG2, ...: registers used to select the special feature. By default, this register is "0" and the inputs are normal (analog value in mv or digital level 0 or 1)
IN_PAR1, IN_PAR2, ... input parameter. In some feature available, the input computation requires a parameter, but most of the available modes just ignore this value.



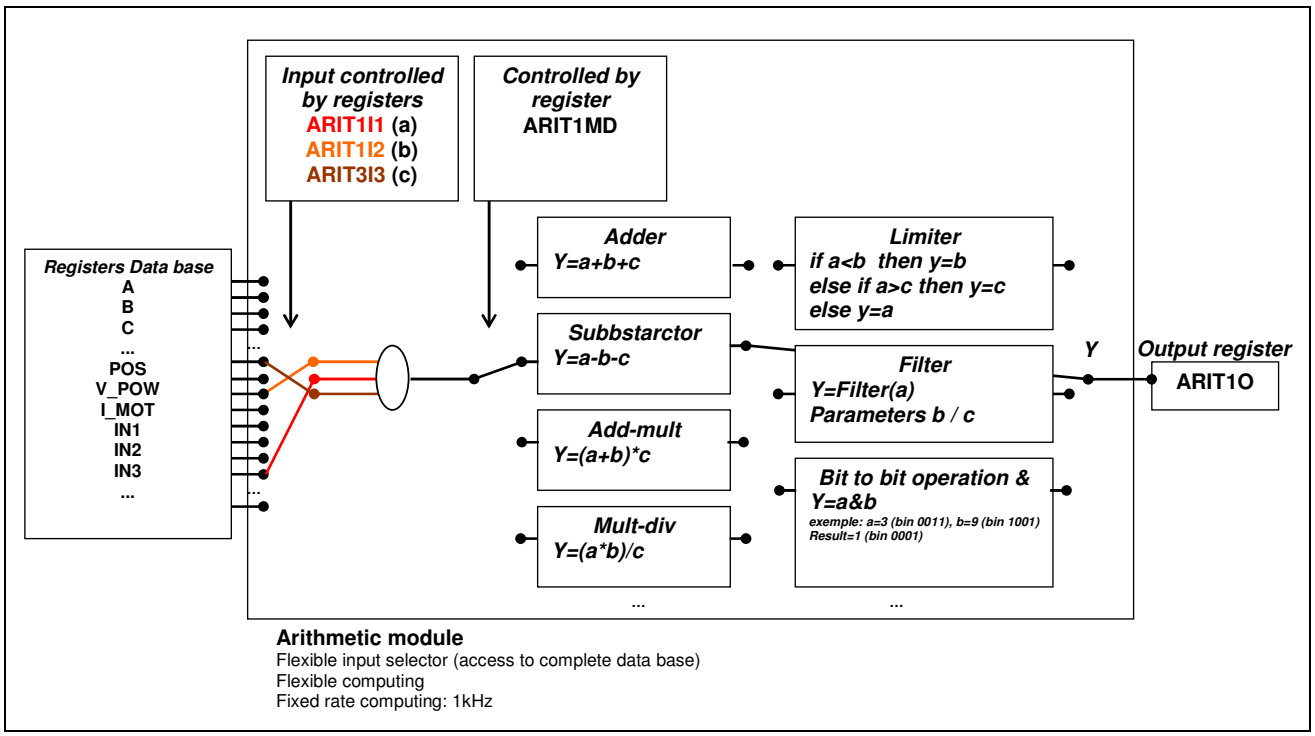
Advanced input module principle.
Registers and pin name is for IN1, the same principle is used for IN2 (pin IN2, register IN_CFG2, IN_PAR2 and IN2), and the other inputs

Please check board specific documentation for details and availability.

Value IN_CFG1	*Description*
0	Y=in (normal mode, register = input voltage or digital level
1 to 31	**analog input**
1	Digitalized, with schmidt trigger, thresholds falling 2.0V / rising 3.5V
2	
3	
32 to 63	**digital input**
32	Digital normal y=in
33	Digital inverted y=1-in
34	
64 to 95	**Movement specific / LED specific**
64	
65	
66	
96 to 127	**High speed timer features**
96	Rizing edges counting UP
97	Rizing edges counting DOWN
98	Falling edges counting UP
99	Falling edges counting DOWN
100	Both edges counting UP
101	Both edges counting DOWN

Arithmetic module

The arithmetic module is designed to be used on signals. For example: changing the scale of an encoder (using Mul-div), or convert 24V analog input into a 0-10V saturated input (using Limiter).



The arithmetic modules are always computed every 1 millisecond.

Compatibility: arithmetic module was introduce in 2010 and is not available on every boards and firmware release. Please check board specific documentation for details.

Arithmetic functions

Value ARITxMD	*Description*
0	OFF
1	ADD y=a+b+c
2	SUB y=a-b-c
3	MUL y=a*b
4	DIV y=a/b
5	MUL DIV relative, with rounding error compensation $Y=y_{-1} + ((a_0-a_{-1})*b)/c$ Ideal for encoder scaling: no rounding error, smooth rolling numbers to manage 32bit capacity exceeding.
11	Saturation Y=a when a is between b and c, otherwize y=b or y=c (the closest)

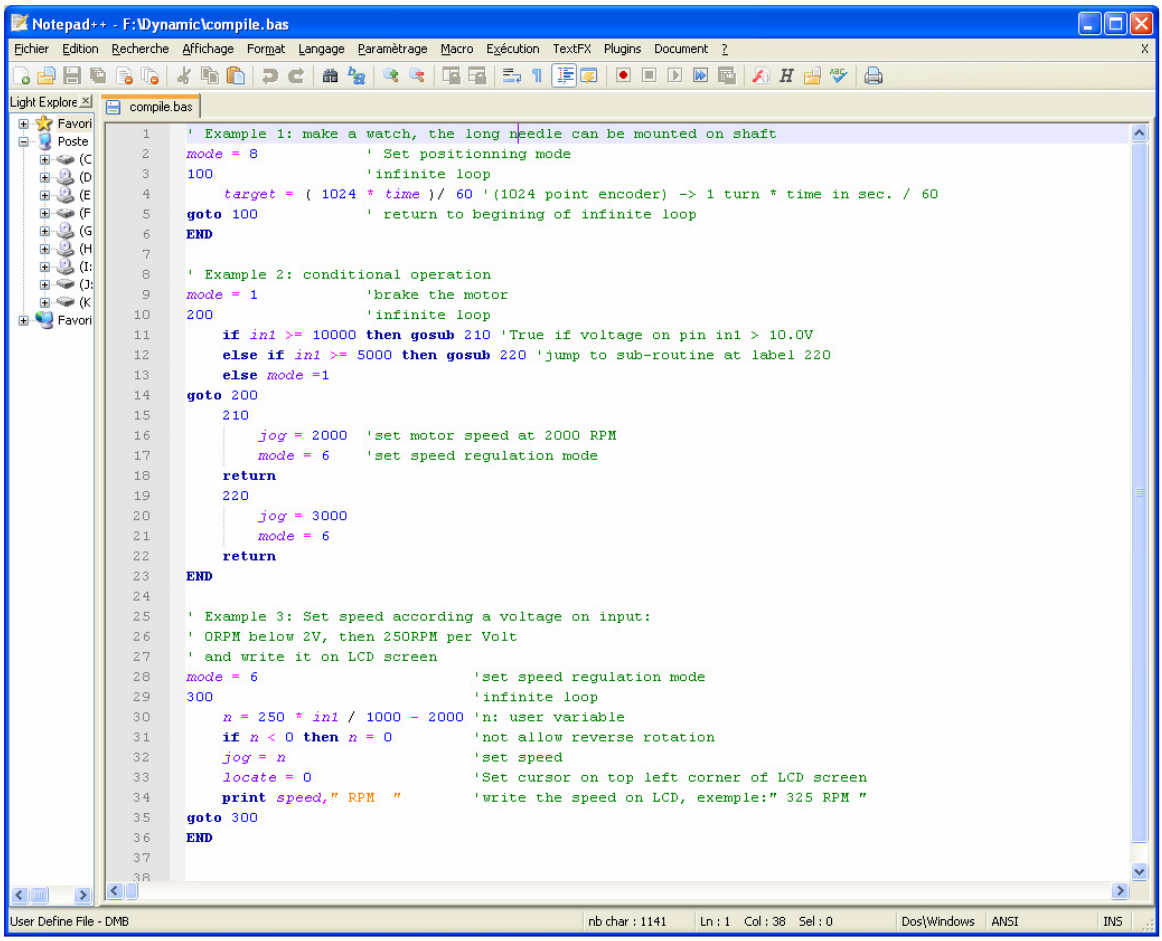
The editor NOTEPAD++

Dynamic Motion recommends using the great editor NOTEPAD++, with the syntax coloration add-on that we provide with.

This NOTEPAD++ editor is free Open source GNU software.

Of course, any other text editor like "Notepad" provided with Windows will work.

When using the color syntax, the programmer has a great help to avoid syntax errors and for a good visibility of the software elements.



Flashing firmware

We normally do not propose firmware update to our customers, however this possibility does exists on almost every models and version.
When an issue is identified, our customers may have the possibility to ask the newest firmware and free software tools to flash themselves the device.

Using the software "Hyper Terminal"

(Software provided with windows)
If for any reason, you prefer using another software in alternative as the one we provide, this section explains how to do it with Hyper Terminal. For other software's, and other platforms (Linux, Mac, ...), similar setup procedure should apply.

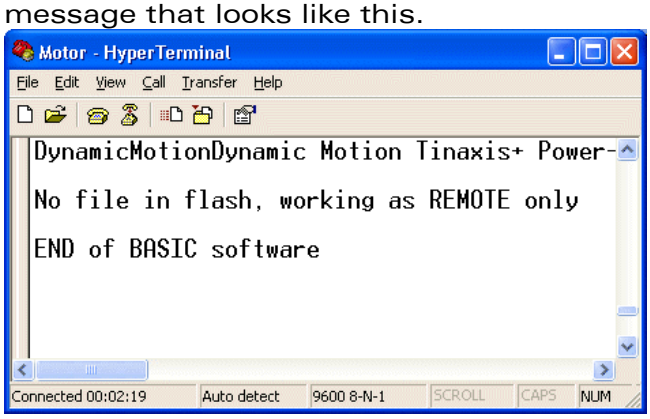
Preparation

Create a new connection with the appropriate parameters:

Give a name to your connection, choose the com port, setup the parameters. (attention, boards with BASIC 2.X uses an speed of 57600)

Use

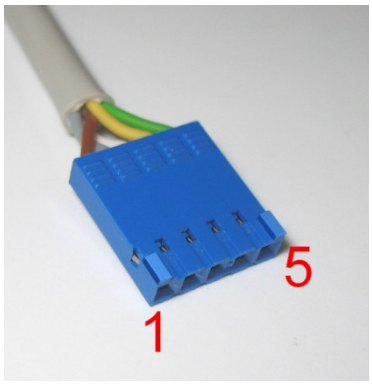
Now you can communicate. To see if it works, at board power-up, you should receive a message that looks like this.



Use the "Dynamic Motion Remote language" to communicate.

To upload your BASIC software, type "ul" (without quotes), then use "send a text file" in the "transfer" menu.
When the file is transferred, the board wait for the character chr28 to end the transfer, so type in the terminal windows Alt 0 2 8 (press and maintain Alt while typing 0 2 8)

Electrical characteristics



Communication with the interface cable:
Pin#

1. +3.3V to 5V (from the board to supply the cable electronic)
2. RX (direction: PC to board)
3. TX (direction: board to PC)
4. GND (0V)
5. Special control input (do not connect)

The connector model is DUOBOX or HE14:
FCI 76384-305LF (on board male connector)
FCI 65240-005LF (cable female connector)

The cable contains a level adapter and inverter, can be based on MAX232 chip.

Connect more than 1 board with the same serial connexion

This is possible thanks to the addressing possibility (software), and require an hardware adaptation to create a kind of bus. The modification are:
- direction computer to boards: connect inputs in parallel
- direction boards to computer: to avoid conflict, a system of pull-up and diode is needed. The passive state is high level, so the configuration of the figure below is required.

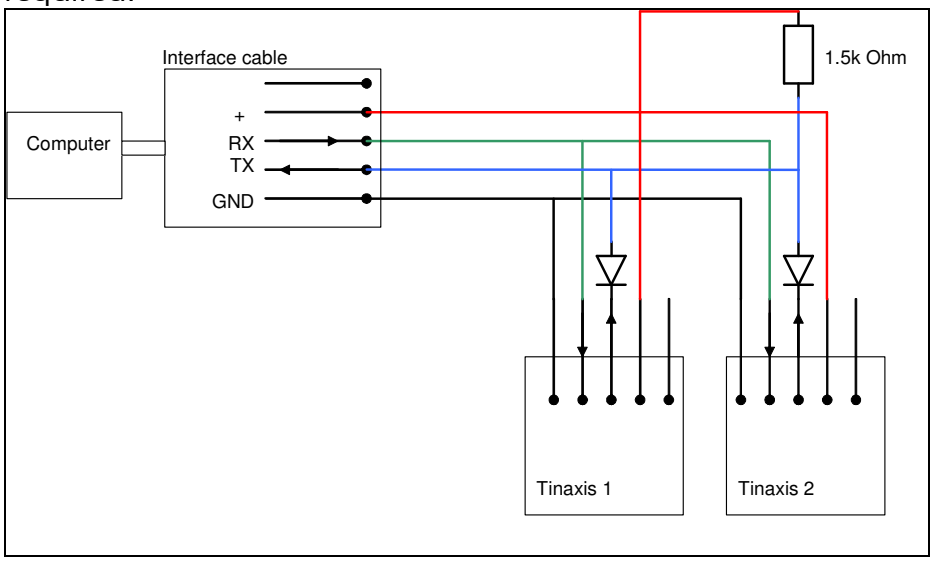


Figure 6, connexions of 2 or more boards on the same RS232 cable

Notes: the length of the cables must be short, and the use of redundancy check is highly recommended.

RS485 BUS principle

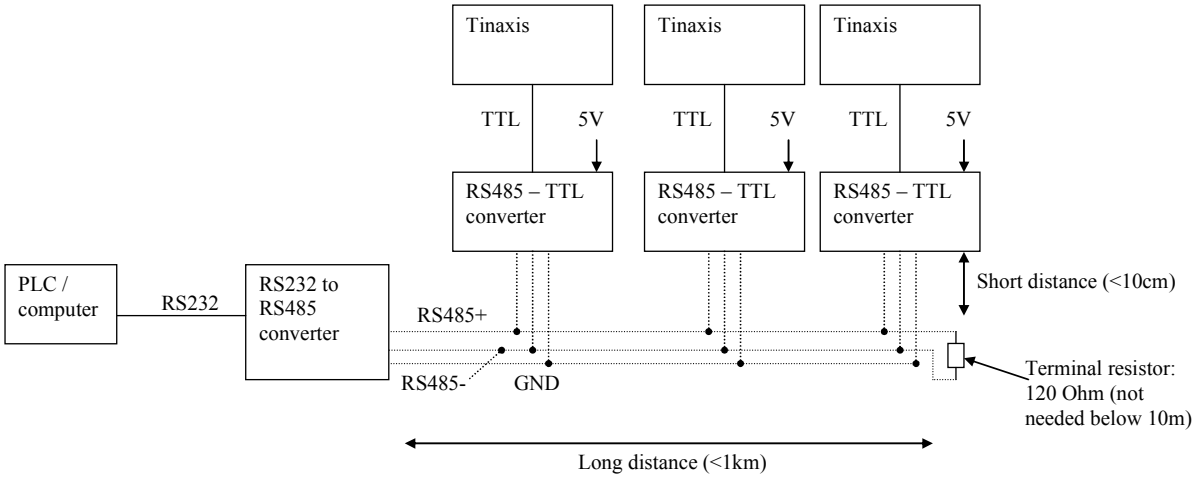


Figure 7, alternative: use RS485 half duplex bus system and auto-timing RS485 converters

Electronic with built-in RS485 serial communication port

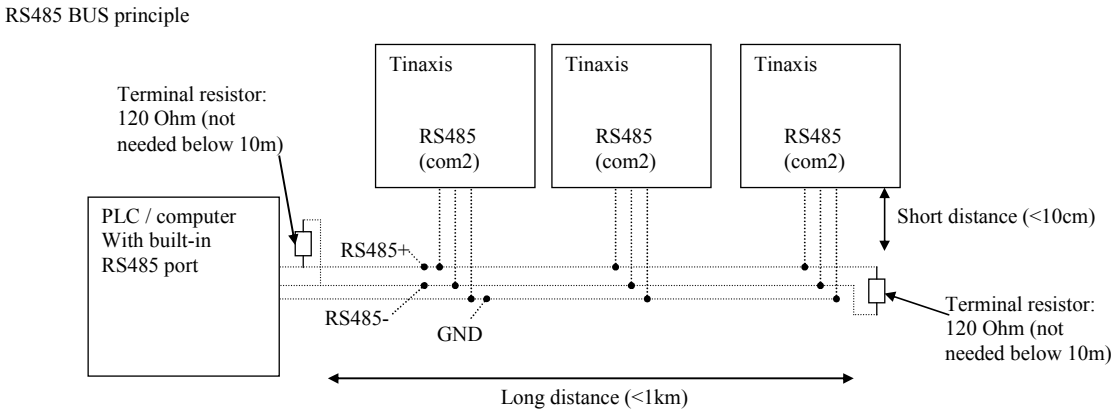


Figure 8, when built-in RS485 port is available, multi-point bus conection is more easy.

Please refer to RS485 / EIA485 specification and recommendation.
RS485 is a half duplex UART serial communication with differential signals.
Shielded twisted pair cable is preferred: shield connected to bus GND, and twisted pair to RS485+ and RS485-.
Converters between USB, RS232, RS485 and TTL can be provided by Dynamic Motion or obtained from many manufacturer.

Closed loop PID Setup

PID loops apply on looped systems, such as BLDC motor with encoder feedback. It does not apply on open loop systems such as stepper motors controllers or LED drivers.

Setting-up a PID is sometimes more tricky than expected. An untuned PID can work very bad, the symptoms are for example:

- Noise (frequency lower than 1kHz)
- Vibration
- Chaotic movement
- Fast movements followed by a stop (over current shut-down)
- Weak rotation
- Too slow or even not moving at all

When the controller is supplied attached to a motor, some basic setups are recorded in the controller, but there is no warranty that the parameters will work for your application

To setup the **PID**, here is an empiric method:

1. Prepare the environment: the motor movements can be quick and sudden. Beware of injury and damages in any motor behavior. If possible, use a protected power supply with a current limitation
2. Write 0 in the derivative factor (PIDx_D), in the integral factor (PIDx_I)
3. Cancel all the feed-forwards (FF_RI=0, FF_SP=0, FF_ACC=0, FF_FRIC=0)
4. Put a low value in the proportional factor PID_P
5. put the motor in tracking mode (MODE=8 or MODE=6)
6. observe the motor movements (play with "TARGET" or "JOG" value to make movements)
7. Tune the PID_P to find the upper limit before vibration. Change the value increasing by 50% each time or reducing by 30% (if value 100 is stable, then try 150, ...)
8. When you have find the limit, reduce the PID_P by 20%

9. Tune the integrator (only if you need to correct offset). Increase PIDx_I using the same method than PIDx_P. When done reduce the value slightly.
10. Then you can tune PIDx_D (it's role is stabilization of low frequency moves). Be careful, derivative factor can introduce vibration.

Then you can tune the **feed-forward** (it's role is to anticipate the moves with the use of simple movement equations. It's highly increase the tracking performance because it reduces the work that the PID has to do). Start by making a small program that represents your movement, and that is repetitive. Example:

```
1 target=1000
pause
target=0
pause
goto 1
```

1. Check and correct the FEM (the BEMF), the R_MOT, the K_MOT and INERTIA values.
2. Upload this program to the motor and test it
3. now put the PID_I, PID_D and PID_P to zero and increase the action of the feed forward until the movement is similar than closed loop: Increase FF_SP and FF_ACC. Normally around 100 for both (100%) the result is the best.
4. Compare the result if you close the loop again (put the right value in PID_P).

When all of this is done, more checks are recommended. If the checks don't pass, then reduce the PID parameters and accept less rigidity in the regulation. This is examples of checks that should be done:

- Use the integrated oscilloscope to verify: the tracking error between POS and MOV_TAR
- Use the oscilloscope to verify the motor current
- Monitor the value PID_COR that show how much work the PID has to do. The lower is the value, the better it is
- Do the tests with the highest and most frequent accelerations your application will do. The PID shows it's limits first in the high changing movements.
- Check the setup will all the inertias, or the biggest and smallest that your application will have
- Check also when your mechanic has wear-out (play, loose belts, ...)
- Be sure the current limitation allows enough power in the motor. (internal limitation: I_MAX and I_C_MAX, and also power supply current)
- Ensure that the energy generated when braking and stabilizing the speed will not perturb the power supply. Especially the switching power supplies are sensitive to this. Monitor the value V_IN (of V_POW is some models) with the integrated oscilloscope and ensure that the voltage is stable. If not, then increase the current consumption taken on the power supply, or change the model, or reduce the deceleration. The boards are protected, it disconnects the motor when voltage is too high, so the motor continues on it inertia until voltage is normal again.

Conclusion: this empiric method is absolutely not the only one, there is mathematical values that works perhaps better, anyway it is more complex to use, require more time to apply and are reserved for people who do this frequently.
For additional information, you can have a look in your favorite book-shop, there is many very good books on the subject.

Contact your Dynamic Motion product seller if you need services for setting-up the PID and related parameters.

Troubleshooting

- BLDC motors / encoder: verify the working of hall sensors and encoder input

Use a software provided with the examples. This will force the digital outputs to the state of the HALL sensors / encoder. The outputs LED will then blink. Rotate the shaft and check the LEDs. Attention, the outputs will not follow the OUTx number. When done, reset the system or write again 0 in DM_CTRL.

- Communication problems: please look at the "Downloading tool" section

Forum, FAQ, Examples

A new forum is available for any question and answer on these products. Please use it.

www.dynamicmotion.ch PRODUCTS FAQ

Examples are available within the environment distribution.

Contact person: Bernard Vaucher (bvaucher@dynamicmotion.ch)